

Learning OWL Class Expressions

Der Fakultät für Mathematik und Informatik
der Universität Leipzig
eingereichte

DISSERTATION

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM
(Dr. rer. nat.)

im Fachgebiet
Informatik

vorgelegt

von **Dipl.-Inf. Jens Lehmann**

geboren am 29. März 1982 in Meissen, Deutschland

Leipzig, den 5.1.2010

Acknowledgement

The thesis was written within the Agile Knowledge Engineering and Semantic Web (AKSW) group hosted by the Chair of Business Information Systems (BIS) at the University of Leipzig. I thank my supervisor Prof. Klaus-Peter Fährnich for granting me the freedom to develop and pursue my research ideas that have lead to this work. I also want to thank Dr. Sören Auer, head of the AKSW group, with whom I had many interesting discussions and who has inspired me with new ideas and suggestions for directions of future work during the past three years. Within our research group I had and have high responsibility with regard to managing projects as well as supervising researchers and students. I also appreciate the great opportunity to work on Semantic Web research with significant impact on practical applications. I want to thank all my colleagues in the AKSW and BIS groups and am glad to be part of these groups. In particular, I thank Sebastian Hellmann for collaborating with me on a broad range of PhD-relevant topics.

Apart from the people already mentioned, draft versions of the thesis were read by Prof. Pascal Hitzler, Prof. Gerhard Brewka, and Prof. Nicola Fanizzi. I thank them for their valuable feedback and support, which helped to improve the quality of the thesis. I am also grateful for previous discussions and work with them as it helped me to acquire and improve important academic skills.

Most of the research ideas described here were implemented and evaluated in the open source DL-Learner and DBpedia projects. I thank everyone working on or using those projects. Specifically, Christoph Haase, Lorenz Bühmann, Christian Kötteritzsch, Steffen Becker, and Sebastian Knappe provided implementations or algorithms directly relevant for this thesis.

Another source of inspiration for the thesis were the Semantic Web and Machine Learning research communities as a whole. During the last years, I met a number of excellent researchers and practitioners in those fields – too many to list them here individually. I am grateful for discussions and research work with them.

Financially, I was mainly supported by the research programmes of the German Ministry for Education and Research and the European Union. Further funding for travel expenses was granted by the German Research Foundation and the German Academic Exchange Service.

Finally, I would like to thank my family and friends for enriching my life beyond my scientific endeavours. In particular, I thank Stephanie for supporting me in hard and stressful times.

Bibliographic Data

Title: Learning OWL Class Expressions

Author: Jens Lehmann

Institution: Universität Leipzig, Fakultät für Mathematik und Informatik

Statistical Information: 223 pages, 35 figures, 23 tables, 3 appendices, 149 literature references, 6 algorithms, 22 examples, 27 definitions, 3 theorems, 17 propositions, 1 corollary, 6 remarks, 11 lemmata, 31 proofs

Abstract

With the advent of the Semantic Web and Semantic Technologies, ontologies have become one of the most prominent paradigms for knowledge representation and reasoning. The popular ontology language OWL, based on description logics, became a W3C recommendation in 2004 and a standard for modelling ontologies on the Web. In the meantime, many studies and applications using OWL have been reported in research and industrial environments, many of which go beyond Internet usage and employ the power of ontological modelling in other fields such as biology, medicine, software engineering, knowledge management, and cognitive systems.

However, recent progress in the field faces a lack of well-structured ontologies with large amounts of instance data due to the fact that engineering such ontologies requires a considerable investment of resources. Nowadays, knowledge bases often provide large volumes of data without sophisticated schemata. Hence, *methods for automated schema acquisition and maintenance* are sought. Schema acquisition is closely related to solving typical classification problems in machine learning, e.g. the detection of chemical compounds causing cancer. In this work, we investigate both, the underlying machine learning techniques and their application to knowledge acquisition in the Semantic Web.

In order to leverage machine-learning approaches for solving these tasks, it is required to develop methods and tools for learning concepts in description logics or, equivalently, class expressions in OWL. In this thesis, it is shown that methods from Inductive Logic Programming (ILP) are applicable to learning in description logic knowledge bases. The results provide foundations for the semi-automatic creation and maintenance of OWL ontologies, in particular in cases when extensional information (i.e. facts, instance data) is abundantly available, while corresponding intensional information (schema) is missing or not expressive enough to allow powerful reasoning over the ontology in a useful way. Such situations often occur

when extracting knowledge from different sources, e.g. databases, or in collaborative knowledge engineering scenarios, e.g. using semantic wikis. It can be argued that being able to learn OWL class expressions is a step towards enriching OWL knowledge bases in order to enable powerful reasoning, consistency checking, and improved querying possibilities. In particular, plugins for OWL ontology editors based on learning methods are developed and evaluated in this work.

The developed algorithms are not restricted to ontology engineering and can handle other learning problems. Indeed, they lend themselves to generic use in machine learning in the same way as ILP systems do. The main difference, however, is the employed knowledge representation paradigm: ILP traditionally uses logic programs for knowledge representation, whereas this work rests on description logics and OWL. This difference is crucial when considering Semantic Web applications as target use cases, as such applications hinge centrally on the chosen knowledge representation format for knowledge interchange and integration. The work in this thesis can be understood as a *broadening of the scope of research and applications of ILP methods*. This goal is particularly important since the number of OWL-based systems is already increasing rapidly and can be expected to grow further in the future.

The thesis starts by establishing the necessary theoretical basis and continues with the specification of algorithms. It also contains their evaluation and, finally, presents a number of application scenarios. The research contributions of this work are threefold:

The first contribution is a *complete analysis of desirable properties of refinement operators in description logics*. Refinement operators are used to traverse the target search space and are, therefore, a crucial element in many learning algorithms. Their properties (completeness, weak completeness, properness, redundancy, infinity, minimality) indicate whether a refinement operator is suitable for being employed in a learning algorithm. The key research question is which of those properties can be combined. It is shown that there is no ideal, i.e. complete, proper, and finite, refinement operator for expressive description logics, which indicates that learning in description logics is a challenging machine learning task. A number of other new results for different property combinations are also proven. The need for these investigations has already been expressed in several articles prior to this PhD work. The theoretical limitations, which were shown as a result of these investigations, provide clear criteria for the design of refinement operators. In the analysis, as few assumptions as possible were made regarding the used description language.

The second contribution is the *development of two refinement operators*. The first operator supports a wide range of concept constructors and it is shown that it is complete and can be extended to a proper operator. It is the most expressive operator designed for a description language so far. The second operator uses the light-weight language \mathcal{EL} and is weakly complete, proper, and finite. It is straightforward to extend it to an ideal operator, if required. It is the first published ideal refinement operator in description logics. While the two operators differ a lot in

their technical details, they both use background knowledge efficiently.

The third contribution is the actual learning algorithms using the introduced operators. New redundancy elimination and infinity-handling techniques are introduced in these algorithms. According to the evaluation, the algorithms produce very readable solutions, while their accuracy is competitive with the state-of-the-art in machine learning. Several optimisations for achieving scalability of the introduced algorithms are described, including a knowledge base fragment selection approach, a dedicated reasoning procedure, and a stochastic coverage computation approach.

The research contributions are evaluated on benchmark problems and in use cases. Standard statistical measurements such as cross validation and significance tests show that the approaches are very competitive. Furthermore, the ontology engineering case study provides evidence that the described algorithms can solve the target problems in practice. A major outcome of the doctoral work is the *DL-Learner framework*. It provides the source code for all algorithms and examples as open-source and has been incorporated in other projects.

Contents

1	Introduction	12
1.1	Motivation	12
1.2	Contributions	14
1.3	Chapter Overview	16
2	Preliminaries and State of the Art	19
2.1	Semantic Web	19
2.1.1	History and Vision	19
2.1.2	RDF and SPARQL	21
2.1.3	Description Logics	23
2.1.4	OWL	31
2.2	Concept Learning and Inductive Reasoning	33
2.2.1	History, Tools, and Applications	35
2.2.2	Learning Problems in OWL/DLs	39
2.2.3	Refinement Operators in OWL/DLs	41
3	Theoretical Foundations of Refinement Operators	45
3.1	The Role of Minimality	45
3.2	Combinations of Completeness, Properness, Finiteness, Redundancy	49
3.3	Weak Completeness	56
4	Designing Refinement Operators	60
4.1	A Complete OWL Refinement Operator	60
4.1.1	Definition of the Operator	61
4.1.2	Completeness of the Operator	65
4.1.3	Achieving Properness	70
4.1.4	Cardinality Restrictions and Concrete Role Support	72
4.1.5	Optimisations	75
4.2	An Ideal EL Refinement Operator	75
4.2.1	\mathcal{EL} Trees and Simulation Relations	77
4.2.2	Formal Description of the \mathcal{EL} Refinement Operator	82
4.2.3	Operator Performance	88
5	Refinement Operator Based OWL Learning Algorithms	90
5.1	OCEL (OWL Class Expression Learner)	90
5.1.1	Redundancy Elimination	90
5.1.2	Creating a Full Learning Algorithm	92

5.2	ELTL (EL Tree Learner)	95
5.3	CELOE (Class Expression Learner for Ontology Engineering) . .	98
6	Improving Scalability of OWL Learning Algorithms	103
6.1	The DBpedia Project	103
6.1.1	The DBpedia Knowledge Extraction Framework	104
6.1.2	The DBpedia Knowledge Base	108
6.1.3	Interlinked Web Content	112
6.1.4	Applications	115
6.2	Knowledge Fragment Selection	119
6.2.1	What Properties Should the Fragment Have?	121
6.2.2	Extending Concise Bound Descriptions (CBDs)	122
6.2.3	Extraction Methods	124
6.2.4	OWL DL Conversion of the Fragment	128
6.2.5	SPARQL Implementation of Tuple Acquisition	128
6.2.6	Usage Scenarios	129
6.3	Optimising Coverage Tests	133
6.3.1	Approximate and Partial Closed World Reasoning	133
6.3.2	Stochastic Coverage Computation	135
7	Implementation, Evaluation, and Use Cases	138
7.1	The DL-Learner Project	138
7.2	ILP Learning Problems	141
7.2.1	Comparison with other Algorithms based on Description Logics	141
7.2.2	Comparison with other ILP approaches	146
7.3	Ontology Engineering	151
7.3.1	The Protégé Plugin	152
7.3.2	The OntoWiki Plugin	154
7.3.3	Evaluation of CELOE	156
7.4	Fragment Extraction Evaluation	159
7.5	Further Applications	163
7.5.1	Predictions of the Effect of Mutations on the Protein Function	163
7.5.2	NLP2RDF	164
7.5.3	ORE - Ontology Repair and Enrichment	164
7.5.4	moosique.net - Music Recommendations	165
7.6	Strengths and Limitations of the Described Approaches	166
8	Related Work	169
8.1	Inductive Learning in Description Logics	169
8.2	Refinement Operators	171
8.3	(Semi-)Automatic Ontology Engineering	172
8.4	Knowledge Fragment Selection	173

9	Conclusions and Future Work	174
9.1	Refinement Operators	174
9.2	Learning Algorithms and Scalability	175
9.3	Implementation, Evaluation and Use Cases	176
9.4	Future Work	177
A	Software Release History	179
B	DL-Learner Manual	180
B.1	What is DL-Learner?	180
B.2	Getting Started	181
B.3	DL-Learner Architecture	182
B.4	DL-Learner Components	184
B.4.1	Knowledge Sources	184
B.4.2	Reasoner Components	185
B.4.3	Learning Problems	186
B.4.4	Learning Algorithms	187
B.5	DL-Learner Interfaces	189
B.6	Extending DL-Learner	190
B.7	General Information	192
C	Curriculum Vitae	193
C.1	Related Peer Reviewed Publications	194
C.2	Other Publications	196
C.3	Talks	197
C.4	Research Projects and Groups	197
C.5	Programm Committee, Reviewing	198
C.6	Seminars and Teaching	199
C.7	Supervision	199
	List of Tables	201
	List of Figures	202
	List of Algorithms	204
	List of Definitions	205
	List of Theorems	206
	List of Examples and Remarks	207
	Bibliography	208
	Selbständigkeitserklärung	223

1 Introduction

With the advent of the Semantic Web and Semantic Technologies, ontologies have become one of the most prominent paradigms for knowledge representation and reasoning. However, recent progress in the field faces a lack of well-structured ontologies with large amounts of instance data due to the fact that engineering such ontologies requires a considerable investment of resources. Nowadays, knowledge bases often provide large volumes of data without sophisticated schemata. Hence, methods for automated schema acquisition and maintenance are sought (see e.g. [Buitelaar et al., 2007]). Schema acquisition is closely related to solving typical classification problems in machine learning, e.g. the detection of chemical compounds causing cancer. In this work, we investigate both, the underlying machine learning techniques and their application to knowledge acquisition in the Semantic Web.

1.1 Motivation

In 2004, the World Wide Web Consortium (W3C) recommended the Web Ontology Language OWL¹ as a standard for modelling ontologies on the Web. In the meantime, many studies and applications using OWL have been reported in research and industrial environments, many of which go beyond Internet usage and employ the power of ontological modelling in other fields such as biology, medicine, software engineering, knowledge management, and cognitive systems [Staab and Studer, 2004, Davies et al., 2006, Hitzler et al., 2009].

In essence, OWL coincides with the description logic *SHOIN*(D) and was extended to support *SRQIQ*(D) [Horrocks et al., 2006] in OWL 2. Description logics (DLs) in general are fragments of first order logic. In order to leverage machine-learning approaches for the creation and extension of OWL ontologies, it is required to develop methods and tools for learning concepts in description logics or, equivalently, class expressions² in OWL. Until recently, only few investigations have been carried out on this topic, which can be attributed to the fact that DLs have only become a widely used paradigm in knowledge representation and reasoning applications since a standard web ontology language was established.

In this thesis, it is shown that methods from Inductive Logic Programming

¹<http://www.w3.org/2004/OWL/>. See also <http://www.w3.org/2007/OWL/> for the currently ongoing revision of the standard.

²http://www.w3.org/TR/owl2-syntax/#Class_Expressions

(ILP) are applicable to learning in description logic knowledge bases³. The results provide foundations for the acquisition of OWL ontologies, in particular in cases when extensional information (facts, instance data) is easily available, while corresponding intensional information (schema) is missing or not expressive enough to allow powerful reasoning over the ontology in a useful way. Such situations often occur when extracting knowledge from different sources, e.g. databases⁴ and wikis [Lehmann et al., 2009], or in collaborative knowledge engineering scenarios, e.g. semantic wikis [Auer et al., 2006a]. The author argues that being able to learn OWL class expressions is a step towards enriching OWL knowledge bases in order to enable powerful reasoning, consistency checking, and improved querying possibilities. In particular, plugins for OWL ontology editors based on the introduced learning methods are developed and evaluated. An example is given below:

Example 1.1 (Simple Ontology Engineering Use Case)

As an example, consider a knowledge base containing a class **Capital** and instances of this class, e.g. London, Paris, Washington, Canberra etc. A machine learning algorithm could suggest that the class **Capital** may be equivalent to one of the following OWL class expressions in Manchester OWL syntax⁵:

City and isCapitalOf min 1 GeopoliticalRegion

City and isCapitalOf min 1 Country

Both suggestions could be plausible: The first one is more general and includes cities that are capitals of states, whereas the latter one is stricter and limits the notion of a capital to countries. A knowledge engineer can decide which one is more appropriate (semi-automatic approach) and the machine learning algorithm should guide her by pointing out which one fits the existing instances better. Assuming the knowledge engineer decides for the second suggestion, an algorithm can show her whether there are instances of the class **Capital** which are not instances of **City** or not related via the property **isCapitalOf** to an instance of **Country**. The knowledge engineer can then continue to look at those instances and assign them to a different class as well as provide more complete information; thus improving the quality of the knowledge base. After adding the definition of **Capital**, an OWL reasoner can compute further instances of the class which have not been explicitly assigned before.

The developed algorithms are, of course, not restricted to ontology engineering and can handle other learning problems. Indeed, they lend themselves to generic use in machine learning in the same way as ILP systems do. The main difference, however, is the employed knowledge representation paradigm: ILP traditionally uses logic programs for knowledge representation, whereas this work rests on DLs/OWL. This difference is crucial when considering Semantic Web applications as target use cases for the presented approaches, as such applications

³Throughout the thesis, the terms knowledge base and ontology are used synonymously.

⁴see e.g. <http://triplify.org>, <http://linkedgeodata.org>

⁵For details on Manchester OWL syntax (e.g. used in Protégé, OntoWiki) see Section 2.1.3 and [Horridge and Patel-Schneider, 2008].

hinge centrally on the chosen knowledge representation format for knowledge interchange and integration. In this respect, the PhD work can be understood as a broadening of the scope of research and applications of ILP methods. This goal is particularly important since the number of OWL-based systems is already increasing rapidly and can be expected to grow further in the future.⁶

A central part in many ILP approaches are so-called *refinement operators* which are used to traverse the search space, and such an ILP-based approach often rests on the definition of a suitable operator. Theoretical investigations on ILP refinement operators have identified desirable properties for them to have, which impact on their performance. These properties, thus, provide general guidelines for the definition of suitable operators, which are not restricted to a logic programming setting, but can be carried over to DLs. It turns out, however, that for expressive DLs there are strong theoretical limitations on the properties a refinement operator can have. A corresponding general analysis, therefore, provides a clear understanding of the difficulties inherent in a learning setting and also allows to derive directions for researching suitable operators.

The author provides both this theoretical analysis and the derivation – including experimental validation – of refinement operators suitable for concept learning in description logics. Concrete refinement operators are given for an expressive fragment of OWL as well as for a light-weight description logic. We show properties of these operators and employ them in three different learning algorithms. While sharing similarities, each of these algorithms is optimised for a specific scenario or refinement operator. Since some knowledge bases in the Semantic Web are very large, particular attention is devoted to scalability. DBpedia, as a specific example of a large collaboratively created knowledge base, is presented. Moreover, two approaches to improve scalability of learning algorithms on large knowledge bases are introduced. An extensive evaluation of the novel learning algorithms is performed by comparing them with other tools for concept learning in description logics and with state-of-the-art machine learning algorithms.

1.2 Contributions

The work in this thesis is cross disciplinary involving Machine Learning and Semantic Web. The Machine Learning part is mainly related to Inductive Logic Programming. It is restricted along three dimensions; thus making it feasible to give the subject a thorough treatment within the bounds of the doctoral thesis:

- supervised (as opposed to unsupervised and reinforcement learning)
- symbolic (as opposed to subsymbolic and non-symbolic)

⁶As a case in point, see the W3C LinkingOpenData Project, <http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData>, <http://linkeddata.org>, or the TONES and Protégé ontology repositories.

- use of DLs/OWL as knowledge representation formalism (as opposed to logic programming, first order logic, datalog etc.)

A detailed explanation of the notion of supervised symbolic learning is given in Section 2.2. In essence, this kind of learning means that a classification problem is solved from given examples by providing logical, human understandable solutions.

The Semantic Web part of the thesis involves knowledge representation (OWL, description logics) as well as application scenarios. In particular, the use of concept learning for ontology engineering is investigated.

The main aim of the thesis is to cover the full spectrum of learning in descriptions logics: It starts by establishing the necessary theoretical basis and continues with the construction of algorithms. It contains an evaluation of them, and, finally, presents application scenarios. The research contributions of this work are threefold:

The first contribution is a full analysis of desirable properties of refinement operators in description logics. These properties (completeness, weak completeness, properness, redundancy, infinity, minimality) indicate whether a refinement operator is suitable for being employed in a learning algorithm. The key research question the author wanted to answer is which of these properties can be combined. When answering this question, a further goal was to make as few assumptions as possible regarding the used description logic. It was found that there is no ideal, i.e. complete, proper, and finite, refinement operator for expressive description logics, which indicates that learning in description logics is a challenging machine learning task. Several other new results for refinement operator property combinations are shown. The need for these investigations has already been expressed in several articles [Fanizzi et al., 2004, Esposito et al., 2004]. The theoretical limitations, which have been found as a result of this work, provide clear criteria for the design of refinement operators.

The second contribution is the development of two refinement operators. The first operator supports a wide range of concept constructors and it is shown that it is complete and can be extended to a proper operator. It is the most expressive operator designed for a description language so far. The second operator uses the light-weight language \mathcal{EL} and is weakly complete, proper, and finite. It is straightforward to extend it to an ideal operator, if required. It is the first ideal refinement operator in description logics to the best of the author's knowledge. While the two operators differ a lot, they both use background knowledge efficiently. The techniques for achieving this are novel.

The third contribution is the actual learning algorithms using the introduced operators. New redundancy elimination and infinity handling techniques in these algorithms are introduced. According to their evaluation, the algorithms produce very readable solutions, while their accuracy is competitive with the state of the art in machine learning. Furthermore, several optimisations for achieving scalability of the introduced algorithms are proposed, including a knowledge base fragment selection approach, dedicated reasoning procedure, and a stochastic

coverage computation approach.

The research contributions are evaluated on benchmark problems and in use cases. Standard statistical measurements such as cross-validation and significance tests show that the approaches are competitive. Furthermore, the ontology engineering use case study provides evidence that the described algorithms can solve the target problems in practice. A major outcome of the doctoral work is the DL-Learner framework. It provides the source code for all algorithms and examples as open-source and has been incorporated in other projects.

The contributions and advancements achieved are listed in more detail in Table 1.1 grouped by their type and including pointers to the relevant part of the document.

1.3 Chapter Overview

Chapter 2 covers the prerequisites necessary to understand the thesis. It offers a brief history of the Semantic Web and highlights key formalisms and technologies with emphasis on description logics and OWL. Similarly, concept learning is described along with typical tools and application examples. Refinement operators and their properties, which play a central role in the definition of the learning algorithms in this thesis, are defined.

Theoretical foundations are laid in Chapter 3. A series of new results on properties of refinement operators in description logics is shown; culminating in Theorem 3.16 which is the central result of this part. Together, the results provide a clear picture of theoretical limitations of properties of refinement operators. To the best of the author's knowledge, such a complete analysis has not been done before. The results provide a foundation for further investigations into practical refinement operators, independent from the rest of this thesis.

Based on these investigations, specific operators are developed in Chapter 4. Apart from proving theoretical properties of these operators, their practical use is discussed as well. In particular, the incorporation of background knowledge in computing refinements plays a central role. While the first operator is designed to cover many features of OWL, the second operator is especially developed for the tractable \mathcal{EL} description logic and has desirable theoretical properties.

The refinement operators are employed in three different learning algorithms (OCEL, ELTL, and CELOE) in Chapter 5. While sharing similarities, each of those algorithms is optimised for a specific scenario or refinement operator.

Since some knowledge bases in the Semantic Web are very large, a dedicated Chapter is devoted to this topic. DBpedia is presented as a specific example of large collaboratively created knowledge base in Section 6.1. It is then shown how the scalability of learning algorithms on large knowledge bases can be improved. The first approach describes the extraction of a relevant fragment from a large knowledge base (Section 6.2) and the second approach shows the efficient execution of coverage tests, in particular in the presence of many objects in a knowledge base.

<i>type</i>	<i>contribution</i>	<i>reference</i>
Theory	thorough investigation of properties of refinement operators for many description languages, including 1.) a non-ideality result for expressive description logics and 2.) an ideality result for \mathcal{EL}	Chapter 3 and Section 4.2
	development of a suitable heuristic for learning class expressions in ontology engineering	Section 5.3
	development of a stochastic method for performing the coverage test of a class expression more efficiently	Section 6.3
Algorithms and Implementation	development and implementation of a refinement operator covering many features of the OWL ontology language	Section 4.1
	development and implementation of a refinement operator for the \mathcal{EL} description logic	Section 4.2
	development of a flexible method for extracting relevant parts of very large and possibly interlinked knowledge bases for a given learning task and its implementation in the DL-Learner framework	Section 6.2
	three learning algorithms (OCEL, ELTL, CELOE) for supervised learning in description logics	Chapter 5
Evaluation	comparison of algorithms on standard ILP Problems with other DL learning approaches and state-of-the-art ILP algorithms	Section 7.2
	evaluation of the CELOE ontology engineering algorithm on several knowledge bases	Section 7.3
	showing computational feasibility of \mathcal{EL} refinement operator on several knowledge bases	Section 4.2.3
Projects, Applications, and Practice	DL-Learner open source project – a framework for supervised learning in OWL and description logics	Section 7.1
	DBpedia open source project – a multi-domain, multi-language knowledge base extraction from Wikipedia	Section 6.1
	Protégé DL-Learner Plugin – a plugin for suggesting class expressions in the popular Protégé ontology editor	Section 7.3.1
	OntoWiki DL-Learner Plugin – a plugin for suggesting class expressions in the web-based OntoWiki platform	Section 7.3.2
	provision of accurate and understandable hypothesis for the carcinogenesis problem	Section 7.2.2
	presentation of application scenarios and examples employing large knowledge bases	Chapter 7

Table 1.1: Contributions and advancements over the state of the art.

Chapter 7 starts with a description of the DL-Learner project. This open source project contains the implementation of all algorithms described as well as a number of examples. It is designed as a framework for learning OWL class expressions and provides several interfaces for developers. The chapter continues with comparing the described learning algorithms and other ILP approaches on challenging problems (Section 7.2). An analysis of the ontology engineering use case (Section 7.3) is made. Two plugins based on DL-Learner for the popular Protégé and OntoWiki ontology editors are presented and evaluated. The scalability improvements presented in Chapter 6 are evaluated and a brief description of further applications is given. Finally, strengths and limitations of the approaches in general are described (Section 7.6).

A summary of related work is presented in Chapter 8. Conclusions and ideas for future work are outlined in Chapter 9.

2 Preliminaries and State of the Art

In this chapter, the basic formalisms and technologies underlying the work in this thesis are introduced. Apart from giving basic notions and concepts, it will also touch on the state of the art in those areas. Expert readers can skip (parts of) this section and continue directly with some novel results obtained during the doctoral work in Chapter 3.

2.1 Semantic Web

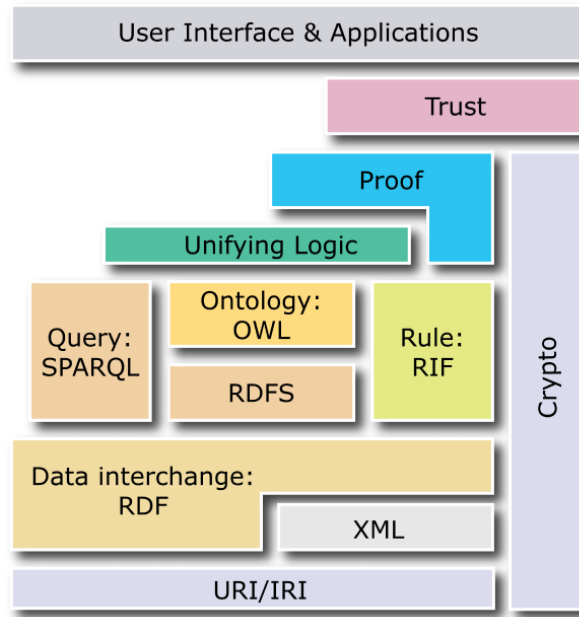
2.1.1 History and Vision

Nowadays, the World Wide Web (WWW) is part of everyday life of many of us and one may have the opinion that the world would be a different place without it. The foundations of the WWW were invented by Tim Berners-Lee in the early 90s. The web has grown rapidly since then. In 1994, only a few years after its invention, Tim Berners-Lee talked about semantics on the web during the first world wide web conference. In this talk¹, he mentions:

“To a computer, then, the web is a flat, boring world devoid of meaning. This is a pity, as in fact documents on the web describe real objects and imaginary concepts, and give particular relationships between them. [...] Adding semantics to the web involves two things: allowing documents which have information in machine-readable forms, and allowing links to be created with relationship values. Only when we have this extra level of semantics will we be able to use computer power to help us exploit the information to a greater extent than our own reading.”

It is interesting to see that this vision of a Semantic Web is almost as old as the web itself. However, it took a few more years until work started towards realising this vision. In 1997, an RDF (Resource Description Framework) working group was initiated within the World Wide Web Consortium (W3C) which led to an official recommendation in 1999. RDF allows to make statements about (web) resources in simple subject-predicate-object triples based on XML and URIs. In 2001, the vision of Tim Berners-Lee and others was described in more detail in the article “The Semantic Web” [Berners-Lee et al., 2001], which is sometimes seen as the launch of broader work on the Semantic Web.

¹<http://www.w3.org/Talks/WWW94Tim/Overview.html>



[Source: <http://www.w3.org/2007/03/layerCake.png>]

Figure 2.1: Semantic Web Layer Cake.

In the following years, a set of W3C standards emerged, which are often summarised in a layer cake (see Figure 2.1). Most notably, the Web Ontology Language (OWL) became a W3C recommendation in 2004. OWL is an expressive knowledge representation language based on description logics and allows powerful reasoning technologies to be applied. In the same year, the RDF specification was revised and RDF Schema (RDFS) became a standard for structuring RDF resources. In 2008, SPARQL was officially announced as W3C standard for querying RDF knowledge bases. One year later, OWL 2, which improves several aspects of OWL, became a W3C recommendation. The set of standards is likely to be extended by the rule interchange format (RIF) in the future.

During this time, the Semantic Web was steadily growing² and contains knowledge from diverse areas such as science, music, people, books, reviews, places, politics, products, software, social networks, as well as upper and general ontologies. The underlying technologies are currently starting to create substantial industrial impact in application scenarios on and off the web, including knowledge management, expert systems, web services, e-commerce, e-collaboration, etc. Despite this partial success, the vision of the Semantic Web as an extension of the World Wide Web has not become a reality yet.

This thesis draws on semantic technologies and describes them in detail in the following sections. As one step towards achieving the Semantic Web vision, it is

²As a rough size estimate, the semantic index Sindice (<http://sindice.com/>) lists more than 10 billion entities from more than 100 million web pages.



[Source: RDF Primer (<http://www.w3.org/TR/rdf-primer/>)]

Figure 2.2: A graph describing Dr. Eric Miller.

shown that the presented learning techniques can simplify the creation of OWL ontologies (detailed in Section 7.3).

2.1.2 RDF and SPARQL

The *Resource Description Framework (RDF)* is a W3C standard for expressing statements about resources. It is used to (globally) identify resources and store knowledge about them in a simple, flexible way, such that it cannot only be viewed by humans, but also processed by applications.

Statements in RDF are stored as subject-predicate-object *triples*, which together form a labeled directed graph. In Figure 2.2, a resource **Eric Miller** is described as a person with the specified name, email address, and title. Resources are represented by URIs, which has the advantage that they are globally identifiable. Resources, which need no global identifier can also be assigned a document local *blank node*. In each triple, the subject is a resource (represented by an ellipse in the graph visualisation), the object is either a resource or a literal (represented by a rectangle), and the predicate/property is an arc from subject to object.

Apart from the graph representation, RDF can be stored in different formats including an XML syntax:

Example 2.1 (RDF/XML Syntax)

```
<?xml version="1.0"?>
```

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

A common non-XML format for RDF is N3³ and its subsets Turtle⁴ and N-Triples⁵. The following represents the above mentioned graph in Turtle syntax:

Example 2.2 (RDF Turtle Syntax)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix contact: <http://www.w3.org/2000/10/swap/pim/contact#>.
```

```
<http://www.w3.org/People/EM/contact#me>
  rdf:type contact:Person;
  contact:fullName "Eric Miller";
  contact:mailbox <mailto:em@w3.org>;
  contact:personalTitle "Dr.".
```

RDF Schema⁶ (RDFS) is a language designed to create RDF vocabularies. Since it is in essence a subset of the OWL ontology language, we refer to Section 2.1.4 for an overview. RDF and RDFS use graph based semantics.

RDF knowledge bases can be made persistent in *triple stores*. *SPARQL*⁷ (SPARQL Query Language for RDF) is the official W3C standard for querying such knowledge bases. It is similar to the SQL language for querying relational databases.

SPARQL is based on the idea of *triple patterns*. A triple pattern is similar to a triple, but each of its parts can also be a variable. A set of triple patterns is a basic graph pattern, which itself can also be viewed as a graph. A basic graph pattern *matches* a subgraph of the RDF knowledge base when RDF terms⁸ in this subgraph can be replaced by variables such that the result is equivalent to the graph of the basic graph pattern.

The following SPARQL query selects all persons with full name “Eric Miller”. The select clause identifies the variables to occur in the results. In this case, `?person` can be replaced by `http://www.w3.org/People/EM/contact#me` to obtain a match.

³<http://www.w3.org/DesignIssues/Notation3.html>

⁴<http://www.w3.org/TeamSubmission/turtle/>

⁵<http://www.w3.org/TR/rdf-testcases/#ntriples>

⁶<http://www.w3.org/TR/rdf-schema/>

⁷<http://www.w3.org/TR/rdf-sparql-query/>

⁸An RDF term is either an IRI (Internationalized Resource Identifier), a literal, or a blank node.

Example 2.3 (SPARQL Query)

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX contact: <http://www.w3.org/2000/10/swap/pim/contact#>
SELECT ?person
WHERE
{
  ?person rdf:type          contact:Person .
  ?person contact:fullName "Eric Miller" .
}

```

FILTERs can be used in SPARQL to restrict solutions to those, which additionally match given expressions. The following query selects persons whose name starts with “Eric”:

Example 2.4 (Filters in SPARQL Queries)

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX contact: <http://www.w3.org/2000/10/swap/pim/contact#>
SELECT ?person
WHERE
{
  ?person rdf:type          contact:Person .
  ?person contact:fullName ?name .
  FILTER regex(?name, "^Eric")
}

```

We refrain from describing RDF and SPARQL in more detail than necessary for understanding this thesis and point the interested reader to the mentioned W3C recommendations⁹ and primers¹⁰ instead.

2.1.3 Description Logics

In this section, we introduce description logics including their syntax and semantics.

Description logics is the name of a family of knowledge representation (KR) formalisms. They emerged from earlier KR formalisms like semantic networks and frames. Their origin lies in the work of Brachman on structured inheritance networks [Brachman, 1978]. Since then, description logics have enjoyed increasing popularity. They can essentially be understood as fragments of first-order predicate logic. They have less expressive power, but usually decidable inference problems and a user-friendly variable free syntax.

Description logics represent knowledge in terms of *objects*, *concepts*, and *roles*. Concepts formally describe notions in an application domain, e.g. one could define the concept of being a father as “a man having a child” ($\text{Father} \equiv \text{Man} \sqcap$

⁹see <http://www.w3.org/RDF/> and <http://www.w3.org/TR/rdf-sparql-query/>

¹⁰<http://www.w3.org/TR/rdf-primer/>

$\exists \text{hasChild}.\top$ in DL notation). Objects are members of concepts in the application domain and roles are binary relations between objects. Objects correspond to constants, concepts to unary predicates, and roles to binary predicates in first-order logic.

In description logic systems information is stored in a *knowledge base*. It is sometimes divided in two parts: *TBox* and *ABox*. The ABox contains *assertions* about objects. It relates objects to concepts and other objects via roles. The TBox describes the *terminology* by relating concepts and roles. For some expressive description logics this clear separation does not exist. Furthermore, the notion of an *RBox*, which contains knowledge about roles, is sometimes used in expressive description logics. We will usually consider those axioms as part of the TBox in this thesis.

As mentioned before, DLs are a family of KR formalisms. We use the terms *description language* and *description logic* synonymously for one particular element of this family. First, we introduce the \mathcal{ALC} description logic as an example language. \mathcal{ALC} is a proper fragment of OWL [Horrocks et al., 2003] and is generally considered to be a prototypical description logic for research investigations. \mathcal{ALC} stands for *attributive language with complement*. It allows to construct complex concepts from simpler ones using various language constructs. The next definition shows how such concepts can be built.

Definition 2.5 (Syntax of \mathcal{ALC} concepts)

Let N_R be a set of *role names* and N_C be a set of *concept names* ($N_R \cap N_C = \emptyset$). The elements of N_C are also called *atomic concepts*. The set of \mathcal{ALC} concepts is inductively defined as follows:

1. Each atomic concept is an \mathcal{ALC} concept.
2. If C and D are \mathcal{ALC} concepts and $r \in N_R$ a role, then the following are also \mathcal{ALC} concepts:
 - \top (top), \perp (bottom)
 - $C \sqcup D$ (disjunction), $C \sqcap D$ (conjunction), $\neg C$ (negation)
 - $\forall r.C$ (value/universal restriction), $\exists r.C$ (existential restriction) □

Example 2.6 (\mathcal{ALC} concepts)

Some examples of complex concepts in \mathcal{ALC} are:

- $\text{Man} \sqcap \exists \text{hasChild}.\top$
- $\text{Man} \sqcap \exists \text{hasChild}.\text{Rich} \sqcup \text{Beautiful}$
- $\text{Man} \sqcap \exists \text{hasChild}.\neg \text{Adult}$
- $\text{Man} \sqcap \exists \text{hasChild}.\forall \text{hasFriend}.\text{ComputerScientist}$ □

Other description languages are usually named according to the expressive features they support. The choice of language is usually a tradeoff between expressivity and complexity of reasoning. The description logic navigator¹¹ provides detailed information about the complexity of a particular language. The following is a list of commonly used letters in the description logic naming scheme along with their meaning (note that if one feature can be expressed using other ones the letter is usually omitted in the language name).

- S \mathcal{ALC} + transitivity: For a transitive role r , we have that $r(a, b)$ and $r(b, c)$ implies $r(a, c)$.
- H subroles: $r \sqsubseteq s$ says that r is a subrole of s , i.e. $r(a, b)$ implies $s(a, b)$.
- I inverse roles: r^- denotes the inverse role of r , i.e. $r^{-1}(a, b)$ iff $r(b, a)$.
- O nominals: Sets of objects can be used to construct concepts, e.g. $\{\text{MONICA}\}$ denotes the singleton set, which only contains MONICA. Nominals are useful in cases where the instances of a concept should be enumerated, e.g. the members of the European Union.
- N number restrictions: Allows constructs of the form $\geq n r$ and $\leq n r$ to build concepts. This is useful if one wants to define a concept like "mother of at least three children" ($\text{Woman} \sqcap \geq 3 \text{ hasChild}$).
- Q qualified number restrictions: Concept constructors of the form $\geq n r.C$ and $\leq n r.C$ can be used. If C is the top concept, this is equivalent to unqualified number restrictions. This is useful to define a concept like "mother of at least three male children" ($\text{Woman} \sqcap \geq 3 \text{ hasChild.Male}$).
- F functional roles: Allows to express that a role r is functional, i.e. has at most one filler, which is equivalent to the axiom $\top \sqsubseteq \leq 1 r$.
- R complex role inclusions: Axioms of the form $r \circ s \sqsubseteq r$ (or $r \circ s \sqsubseteq s$) state that when $r(a, b)$ and $s(b, c)$ holds, then $r(a, c)$ (or $s(a, c)$) also holds. For instance, we could use the axiom $\text{locatedIn} \circ \text{partof} \sqsubseteq \text{locatedIn}$ to model the part of relationship for locations. Now, if we know that Leipzig is located in Saxony and Saxony is part of Germany, we can infer that Leipzig is located in Germany.
- D data types: Data types are used to incorporate different kinds of data, e.g. numbers or strings. This allows, for instance, to define the concept of an old person as a person of age 65 or higher.

¹¹<http://www.cs.manchester.ac.uk/~ezolin/dl/>

construct	syntax	semantics
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
nominals	$\{o\}$	$\{o\}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}, \{o\}^{\mathcal{I}} = 1$
top concept	\top	$\Delta^{\mathcal{I}}$
bottom concept	\perp	\emptyset
conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
exists restriction	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{a \mid \exists b.(a, b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$
value restriction	$\forall r.C$	$(\forall r.C)^{\mathcal{I}} = \{a \mid \forall b.(a, b) \in r^{\mathcal{I}} \text{ implies } b \in C^{\mathcal{I}}\}$
atleast restriction	$\geq n \ r.C$	$(\geq n \ r)^{\mathcal{I}} = \{a \mid (\{b \mid (a, b) \in r^{\mathcal{I}}\}) \geq n\}$
atmost restriction	$\leq n \ r.C$	$(\leq n \ r)^{\mathcal{I}} = \{a \mid (\{b \mid (a, b) \in r^{\mathcal{I}}\}) \leq n\}$

 Table 2.1: Syntax and semantics for concepts in \mathcal{SHOIN} .

While \mathcal{ALC} is seen as a prototypical language and foundation for more expressive languages, there has also been a lot of research effort for simple languages with often tractable inference problems. Two of those languages, which are referred to within the thesis are \mathcal{AL} and \mathcal{EL} :

\mathcal{AL} is inductively defined as follows: $\top, \perp, \exists r.\top, A, \neg A$ with $A \in N_C, r \in N_R$ are \mathcal{AL} concepts. If C and D are \mathcal{AL} concepts, then $C \sqcap D$ is an \mathcal{AL} concept. If C is an \mathcal{AL} concept and r a role, then $\forall r.C$ is an \mathcal{AL} concept.

\mathcal{EL} is inductively defined as follows: \top, A with $A \in N_C$ are \mathcal{EL} concepts. If C and D are \mathcal{EL} concepts and $r \in N_R$, then $C \sqcap D$ and $\exists r.C$ are \mathcal{EL} concepts.

The semantics of concepts is defined by means of interpretations. See the following definition and Table 2.1 listing common concept constructors.

Definition 2.7 (Interpretation)

An *interpretation* \mathcal{I} consists of a non-empty *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$, which assigns to each $A \in N_C$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to each $r \in N_R$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. \square

Example 2.8 (Interpreting Concepts)

Let the interpretation \mathcal{I} be given by:

$$\begin{aligned} \Delta^{\mathcal{I}} &= \{\text{MONICA, JESSICA, STEPHEN}\} \\ \text{Woman}^{\mathcal{I}} &= \{\text{MONICA, JESSICA}\} \\ \text{hasChild}^{\mathcal{I}} &= \{(\text{MONICA, STEPHEN}), (\text{STEPHEN, JESSICA})\} \end{aligned}$$

We then have:

$$(\text{Woman} \sqcap \exists \text{hasChild}.\top)^{\mathcal{I}} = \{\text{MONICA}\}$$

\square

In the most general case, *terminological axioms* are of the form $C \sqsubseteq D$ or $C \equiv D$, where C and D are (complex) concepts. The former axioms are called *inclusions* and the latter *equivalences*. An equivalence whose left hand side is an atomic concept is a *concept definition*. In some languages with low expressivity, like \mathcal{AL} , terminological axioms are restricted to definitions. We can define the semantics of terminological axioms in a straightforward way. An interpretation \mathcal{I} satisfies an inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and it satisfies the equivalence $C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. \mathcal{I} satisfies a set of terminological axioms iff it satisfies all axioms in the set. An interpretation, which satisfies a (set of) terminological axiom(s) is called a *model* of this (set of) axiom(s). Two (sets of) axioms are *equivalent* if they have the same models. A finite set \mathcal{T} of terminological axioms is called a (*general*) *TBox*. Let N_I be the set of object names (disjoint with N_R and N_C). An *assertion* has the form $C(a)$ (*concept assertion*), $r(a, b)$ (*role assertion*), where a, b are object names, C is a concept, and r is a role. An *ABox* \mathcal{A} is a finite set of assertions.

Objects are also called individuals. To allow interpreting ABoxes we extend the definition of an interpretation. In addition to mapping concepts to subsets of our domain and roles to binary relations, an interpretation has to assign to each individual name $a \in N_I$ an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation \mathcal{I} is a model of an ABox \mathcal{A} (written $\mathcal{I} \models \mathcal{A}$) iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all $C(a) \in \mathcal{A}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ for all $r(a, b) \in \mathcal{A}$. An interpretation \mathcal{I} is a model of a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ (written $\mathcal{I} \models \mathcal{K}$) iff it is a model of \mathcal{T} and \mathcal{A} .

Example 2.9 (Models of a Knowledge Base)

Let the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be given by:

TBox \mathcal{T} :

$$\begin{aligned} \text{Man} &\equiv \neg \text{Woman} \sqcap \text{Person} \\ \text{Woman} &\sqsubseteq \text{Person} \\ \text{Mother} &\equiv \text{Woman} \sqcap \exists \text{hasChild}.\top \end{aligned}$$

ABox \mathcal{A} :

$$\begin{aligned} &\text{Man}(\text{STEPHEN}). \\ &\neg \text{Man}(\text{MONICA}). \\ &\text{Woman}(\text{JESSICA}). \\ &\text{hasChild}(\text{STEPHEN}, \text{JESSICA}). \end{aligned}$$

We will now look at some interpretations and determine whether or not they are a model of \mathcal{K} . For all interpretations, the domain $\{\text{MONICA}, \text{JESSICA}, \text{STEPHEN}\}$ is used and all object names are interpreted in the obvious way (STEPHEN is interpreted as STEPHEN etc.).

Let the interpretation \mathcal{I}_1 be given by:

$$\begin{aligned}\text{Man}^{\mathcal{I}_1} &= \{\text{JESSICA}, \text{STEPHEN}\} \\ \text{Woman}^{\mathcal{I}_1} &= \{\text{MONICA}, \text{JESSICA}\} \\ \text{Mother}^{\mathcal{I}_1} &= \emptyset \\ \text{Person}^{\mathcal{I}_1} &= \{\text{JESSICA}, \text{MONICA}, \text{STEPHEN}\} \\ \text{hasChild}^{\mathcal{I}_1} &= \{(\text{STEPHEN}, \text{JESSICA})\}\end{aligned}$$

Clearly this does not satisfy \mathcal{T} , because the definition $\text{Man} \equiv \neg \text{Woman} \sqcap \text{Person}$ is not satisfied. We have $\text{Man}^{\mathcal{I}_1} = \{\text{JESSICA}, \text{STEPHEN}\}$ and $(\neg \text{Woman} \sqcap \text{Person})^{\mathcal{I}_1} = \{\text{STEPHEN}\}$, which are not equal. However, \mathcal{I}_1 satisfies \mathcal{A} .

Let the interpretation \mathcal{I}_2 be given by:

$$\begin{aligned}\text{Man}^{\mathcal{I}_2} &= \{\text{STEPHEN}\} \\ \text{Woman}^{\mathcal{I}_2} &= \{\text{JESSICA}, \text{MONICA}\} \\ \text{Mother}^{\mathcal{I}_2} &= \emptyset \\ \text{Person}^{\mathcal{I}_2} &= \{\text{JESSICA}, \text{MONICA}, \text{STEPHEN}\} \\ \text{hasChild}^{\mathcal{I}_2} &= \emptyset\end{aligned}$$

\mathcal{I}_2 satisfies \mathcal{T} , but not \mathcal{A} . We have $\text{hasChild}(\text{STEPHEN}, \text{JESSICA}) \in \mathcal{A}$, but $(\text{STEPHEN}^{\mathcal{I}_2}, \text{JESSICA}^{\mathcal{I}_2}) \notin \text{hasChild}^{\mathcal{I}_2}$.

Let the interpretation \mathcal{I}_3 be given by:

$$\begin{aligned}\text{Man}^{\mathcal{I}_3} &= \{\text{STEPHEN}\} \\ \text{Woman}^{\mathcal{I}_3} &= \{\text{JESSICA}, \text{MONICA}\} \\ \text{Mother}^{\mathcal{I}_3} &= \{\text{MONICA}\} \\ \text{Person}^{\mathcal{I}_3} &= \{\text{JESSICA}, \text{MONICA}, \text{STEPHEN}\} \\ \text{hasChild}^{\mathcal{I}_3} &= \{(\text{MONICA}, \text{STEPHEN}), (\text{STEPHEN}, \text{JESSICA})\}\end{aligned}$$

\mathcal{I}_3 is a model of \mathcal{T} and \mathcal{A} , so it is a model of \mathcal{K} . One may argue that nothing in our knowledge base justifies the fact that we interpret **MONICA** as mother. However, in DLs we usually have the *open world assumption*. This means that the given knowledge is viewed as incomplete. There is nothing, which tells us that **MONICA** is not a mother. In databases one usually uses the *closed world assumption*, i.e. all facts, which are not explicitly stored, are assumed to be false. \square

As we have described, a knowledge base can be used to represent the information we have about an application domain. Besides this *explicit* knowledge, we can also deduce *implicit* knowledge from a knowledge base. It is the aim of *inference algorithms* to extract such implicit knowledge. There are some standard reasoning tasks in description logics, which we will briefly describe.

In *terminological reasoning* we reason about concepts. The standard problems are *consistency*, *satisfiability* and *subsumption*. Intuitively, consistency checks detect whether a knowledge base contains contradictions. Satisfiability determines whether a concept can be satisfied, i.e. it is free of contradictions. Subsumption of two concepts detects whether one of the concepts is more general than the other.

Definition 2.10 (Consistency)

A knowledge base \mathcal{K} is *consistent* iff it has a model. □

Example 2.11 (Consistency)

The knowledge base $\mathcal{K} = \{A_1 \equiv A_2 \sqcap \neg A_2, A_1(a)\}$ is not consistent, since A_1 is equivalent to \perp and has an asserted instance a . □

Definition 2.12 (Satisfiability)

Let C be a concept and \mathcal{T} a TBox. C is *satisfiable* iff there is an interpretation \mathcal{I} such that $C^{\mathcal{I}} \neq \emptyset$. C is *satisfiable with respect to \mathcal{T}* iff there is a model \mathcal{I} of \mathcal{T} such that $C^{\mathcal{I}} \neq \emptyset$. □

Example 2.13 (Satisfiability)

$\text{Man} \sqcap \text{Woman}$ is satisfiable. However, it is not satisfiable with respect to the TBox in Example 2.9. □

Definition 2.14 (Subsumption, Equivalence)

Let C, D be concepts and \mathcal{T} a TBox. C is *subsumed by D* , denoted by $C \sqsubseteq D$, iff for any model \mathcal{I} we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. C is *subsumed by D with respect to \mathcal{T}* , denoted by $C \sqsubseteq_{\mathcal{T}} D$, iff for any model \mathcal{I} of \mathcal{T} we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

C is *equivalent to D (with respect to \mathcal{T})*, denoted by $C \equiv D$ ($C \equiv_{\mathcal{T}} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_{\mathcal{T}} D$) and $D \sqsubseteq C$ ($D \sqsubseteq_{\mathcal{T}} C$).

C is *strictly subsumed by D (with respect to \mathcal{T})*, denoted by $C \sqsubset D$ ($C \sqsubset_{\mathcal{T}} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_{\mathcal{T}} D$) and not $C \equiv D$ ($C \equiv_{\mathcal{T}} D$). □

Example 2.15 (Subsumption)

Mother is not subsumed by Woman . However, Mother is subsumed by Woman with respect to the TBox in Example 2.9. □

Subsumption allows to build a hierarchy of atomic concepts, commonly called the *subsumption hierarchy*. Analogously, for more expressive description logics *role hierarchies* can be inferred.

In *assertional reasoning* one reasons about objects. As one relevant task for learning in DLs, the *instance check problem* is to find out whether an object is an instance of a concept, i.e. belongs to it. A *retrieval* operation finds all instances of a given concept.

Definition 2.16 (Instance Check)

Let \mathcal{A} be an ABox, \mathcal{T} a TBox, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ a knowledge base, C a concept, and $a \in N_I$ an object. a is an *instance of C with respect to \mathcal{A}* , denoted by $\mathcal{A} \models C(a)$,

iff in any model \mathcal{I} of \mathcal{A} we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$. a is an instance of C with respect to \mathcal{K} , denoted by $\mathcal{K} \models C(a)$, iff in any model \mathcal{I} of \mathcal{K} we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$.

To denote that a is not an instance of C with respect to \mathcal{A} (\mathcal{K}) we write $\mathcal{A} \not\models C(a)$ ($\mathcal{K} \not\models C(a)$). \square

We use the same notation for sets S of assertions of the form $C(a)$, e.g. $\mathcal{K} \models S$ means that every element in S follows from \mathcal{K} .

Definition 2.17 (Retrieval)

Let \mathcal{A} be an ABox, \mathcal{T} a TBox, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ a knowledge base, C a concept. The *retrieval* $R_{\mathcal{A}}(C)$ of a concept C with respect to \mathcal{A} is the set of all instances of C : $R_{\mathcal{A}}(C) = \{a \mid a \in N_I \text{ and } \mathcal{A} \models C(a)\}$. Similarly the *retrieval* $R_{\mathcal{K}}(C)$ of a concept C with respect to \mathcal{K} is $R_{\mathcal{K}}(C) = \{a \mid a \in N_I \text{ and } \mathcal{K} \models C(a)\}$. \square

Example 2.18 (Instance Check, Retrieval)

In Example 2.9 we have $R_{\mathcal{K}}(\text{Woman}) = \{\text{JESSICA}, \text{MONICA}\}$. JESSICA and MONICA are instances of Woman, because in any model \mathcal{I} of \mathcal{K} we have $\text{JESSICA}^{\mathcal{I}} \in \text{Woman}^{\mathcal{I}}$ and $\text{MONICA}^{\mathcal{I}} \in \text{Woman}^{\mathcal{I}}$. \square

We introduce some further notions, which are used in the thesis. A concept is in *negation normal form* iff negation only occurs in front of concept names. The *length* of a concept is defined in a straightforward way, namely as the sum of the numbers of concept names, role names, quantifier, and connective symbols occurring in the concept. In particular, for \mathcal{ALC} we have:

Definition 2.19 (Length of an \mathcal{ALC} Concept)

The *length* $|C|$ of a concept C is defined inductively (A stands for an atomic concept):

$$\begin{aligned} |A| &= |\top| = |\perp| = 1 \\ |\neg D| &= |D| + 1 \\ |D \sqcap E| &= |D \sqcup E| = 1 + |D| + |E| \\ |\exists r.D| &= |\forall r.D| = 2 + |D| \end{aligned} \quad \square$$

The *depth* of a concept is the maximal number of nested concept constructors. The *role depth* of a concept is the maximal number of nested roles. A *subconcept* of a concept C is a concept syntactically contained in C . For brevity we sometimes omit brackets. In this case, constructors involving quantifiers have higher priority, e.g. $\exists r.\top \sqcap A$ means $(\exists r.\top) \sqcap A$. In several proofs in the thesis we use a convenient abbreviated notation to denote $\forall r$ chains and $\exists r$ chains:

$$\forall r^n = \underbrace{\forall r \dots \forall r}_{n\text{-times}} \quad \exists r^n = \underbrace{\exists r \dots \exists r}_{n\text{-times}}$$

We refer the interested reader to [Horrocks et al., 2006, Baader et al., 2007a, Hitzler et al., 2009] for more detailed information about description logics.

2.1.4 OWL

After we have introduced description logics, we will now describe their relationship to OWL (Web Ontology Language). In essence OWL is based on description logics extended by several features to make it suitable as a web ontology language, e.g. using URIs/IRIs as identifiers, imports of other ontologies etc. By basing OWL-DL on description logics, it can make use of the theory developed for DLs, in particular sophisticated reasoning algorithms.

In OWL, different naming conventions are used compared to description logics. OWL *classes* correspond to concepts in description logics and *properties* correspond to roles.

OWL comes in three flavors: OWL Lite, OWL DL, and OWL Full. OWL Lite corresponds to $\mathcal{SHIF}(D)$ and OWL DL to $\mathcal{SHOIN}(D)$. OWL Full contains features not expressible in description logics, but needed to be compatible with RDFS, i.e. OWL Full can be seen as the union of RDFS and OWL DL.

The latest version OWL 2 is again split in two flavors OWL 2 DL and OWL 2 Full. OWL 2 DL corresponds to the logic $\mathcal{SROIQ}(D)$, whereas the full variant is again introduced for RDFS compatibility. In addition, three profiles were introduced: EL, QL, and RL. Each profile imposes, usually syntactical, restrictions on OWL in order to allow more efficient reasoning. OWL 2 EL is aimed at applications which require expressive property modelling and is based on the logic $\mathcal{EL}++$, which guarantees polynomial reasoning time wrt. ontology size for all standard inference problems. QL is targeted at applications with massive volumes of instance data. In QL, query answering can be implemented on top of conventional relational database systems and sound and complete conjunctive query answering methods can be implemented in LOGSPACE. As in the EL profile, the standard inference problems run in polynomial time. RL is aimed at scalable applications, which however, do not want to sacrifice too much expressive power. Reasoning algorithms for it can be implemented in rule-based engines and run in polynomial time. The EL and QL languages are subsets of OWL 2 DL, whereas RL provides two variants where one is subset of OWL 2 Full and the other one is a subset of OWL 2 DL.

In general, OWL offers more convenience constructs than the corresponding description logics, but does not extend its expressivity. It should be noted that OWL does not make the unique name assumption, so different individuals can be mapped to the same domain element. It allows to express equality and inequality between individuals ($a = b$, $a \neq b$) using `owl:sameAs` and `owl:differentFrom`. Most algorithms for description logics already supported this before the OWL specification was created. Not making the unique names assumption is crucial in the Semantic Web, where it is often the case that many knowledge bases contain information about the same entity. In this case, a common approach is that each knowledge base uses their own URI and `owl:sameAs` is used to connect them (see the information about DBpedia in Section 6.1).

Table 2.2 shows for some examples how constructs in OWL can be mapped

OWL expression / axiom	DL syntax	Manchester syntax
Thing	\top	Thing
Nothing	\perp	Nothing
intersectionOf	$C_1 \sqcap \dots \sqcap C_n$	C_1 and ... and C_n
unionOf	$C_1 \sqcup \dots \sqcup C_n$	C_1 or ... or C_n
complementOf	$\neg C$	not C
oneOf	$\{x_1\} \sqcup \dots \sqcup \{x_n\}$	$\{x_1, \dots, x_n\}$
allValuesFrom	$\forall r. C$	r only C
someValuesFrom	$\exists r. C$	r some C
maxCardinality	$\leq n \ r$	r max n
minCardinality	$\geq n \ r$	r min n
cardinality	$\leq n \ r \sqcap \geq n \ r$	r exact n
subClassOf	$C_1 \sqsubseteq C_2$	C_1 SubClassOf: C_2
equivalentClass	$C_1 \equiv C_2$	C_1 EquivalentTo: C_2
disjointWith	$C_1 \equiv \neg C_2$	C_1 DisjointWith: C_2
sameAs	$\{x_1\} \equiv \{x_2\}$	x_1 SameAs: x_2
differentFrom	$\{x_1\} \sqsubseteq \neg \{x_2\}$	x_1 DifferentFrom: x_2
domain	$\forall r. \top \sqsubseteq C$	r Domain: C
range	$\top \sqsubseteq \forall r. C$	r Range: C
subPropertyOf	$r_1 \sqsubseteq r_2$	r_1 SubPropertyOf: r_2
equivalentProperty	$r_1 \equiv r_2$	r_1 EquivalentTo: r_2
inverseOf	$r_1 \equiv r_2^-$	r_1 InverseOf: r_2
TransitiveProperty	$r^+ \sqsubseteq r$	r Characteristics: Transitive
FunctionalProperty	$\top \sqsubseteq \leq 1 \ r$	r Characteristics: Functional

Table 2.2: OWL constructs in DL and Manchester OWL syntax (excerpt).

to description logics. We can see that some features can be mapped directly to description logics, e.g. union, and others are syntactic sugar, e.g. functional properties.

OWL also has different syntactic formats, in which a knowledge base can be stored. Since it can be converted to RDF, formats like RDF/XML or Turtle can be used. There is also a special XML syntax called OWL/XML and the Manchester OWL Syntax. The latter one is popular in ontology editors. Examples are shown on the right column in Table 2.2.

2.2 Concept Learning and Inductive Reasoning

In the previous section, we have introduced Semantic Web technologies and the underlying knowledge representation standards. The second major research area relevant for this thesis is Machine Learning. Before we delve deeper into the considered learning problems we define related notions. To make the notions easy to understand, we will first give a broad overview and then move towards concept learning in descriptions logics.

Machine Learning in general is a subfield of Artificial Intelligence. The main goal is to develop algorithms that allow computers to improve their performance on a given goal with more data. Often, the ability to learn is seen as a sign or even prerequisite for intelligent behaviour. According to [Mitchell, 1997] “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .”. This is a very general description covering diverse tasks. For instance, in speech recognition, a possible task T is to recognize spoken words. The performance measure P could be the percentage of correctly recognized words. Experience E could consist of existing mappings between spoken and written words. According to the definition above, a learner improves its performance with more experience. In this case, this means that it has a better capability of recognizing spoken words when more mappings are provided.

Machine learning problems and algorithms can be divided in several groups: One distinction is that between supervised, unsupervised, and reinforcement learning methods.

In *supervised learning*, input-output training examples are supplied by a trainer or oracle. Given this training data, the learner tries to find a function mapping between input and output. The output can be a continuous value – a so called *regression* problem – or a class label – a so called *classification* problem. For instance, finding an underlying curve given a set of points is a regression problem whereas detecting whether an email is spam or not is a classification problem.

Unsupervised learning problems are those where training data is not labelled and the aim is to detect the underlying structure of the data. A typical example is clustering. Considering a social network of several people, clustering algorithms can be used to find relevant groups within the social network.

A third category of machine learning techniques is *reinforcement learning*. In this case, an environment is considered which can be changed by actions of the learner. Depending on the consequences of an action, the learner receives reward or punishment as feedback. Using this feedback, the learner can improve its performance by choosing appropriate actions given a certain state of the environment. Reinforcement learning has been applied to robot control and game playing.

Another distinction is that between symbolic, sub-symbolic, and non-symbolic approaches. Symbolic methods produce human-readable results, e.g. logical formulas or decision trees, whereas non-symbolic methods like neural networks are not considered to be human understandable, since they cannot provide a suc-

cinct explanation of their behaviour. In between those extremes are sub-symbolic methods, e.g. Bayesian approaches. As an example, a Bayesian spam filter cannot give a reasonably short and complete explanation why it classifies emails as spam. However, it can give a list of words which are important indicators for spam emails.

In this thesis, we are concerned with *Inductive Learning from Examples*, which is a subfield of symbolic, supervised Machine Learning. The word induction means to infer general principles from specific facts or instances. Sometimes, inductive learning is also called *inductive reasoning* as opposed to deductive reasoning. In Inductive Learning, the general principles are expressed in some logical language, for instance first order logic, logic programs, or description logics. *Inductive Logic Programming* (ILP) has been an active area of research over more than 15 years. Often, ILP is viewed as the intersection of induction and logic programming, i.e. $ILP = I \cap LP$ [Lavrac and Dzeroski, 1994]. The content of this thesis is closely related to ILP, but considers description logics as knowledge representation formalism instead.

(Inductive) *concept learning* is a more specific problem setting than inductive learning. The aim is to find a logical description of a concept from given members (and non-members) of this concept. In this thesis, the notion of a concept can be understood as (atomic) description logic concept exactly as defined in Section 2.1.3. Naturally, “concepts” exist in other knowledge representation languages as well. A logical description of a concept, which arises during a learning process, is called a *hypothesis*, since it is a tentative explanation of why the objects are members (or non-members) of the concept. Members of a concept are called *positive examples* and non-members are *negative examples*. If an example belongs to a hypothesis, i.e. we can infer that it is an instance of the hypothesis, we say the hypothesis *covers* the example. A hypothesis is *complete* if it covers all positive examples. Otherwise, it is called *too weak*. A hypothesis is *consistent* if it does not cover any negative example and is called *too strong* otherwise. A complete and consistent hypothesis is *correct*. We say that a hypothesis is *overly general* if it is complete but not consistent. It is *overly specific* if it consistent, but not complete.

A characteristic feature of most inductive learning approaches is the use of *background knowledge*. This allows more complex learning scenarios, since not only the factual description of the given examples can be used by the learner, but rich knowledge in an appropriate representation language can be taken into account. For this reason, inductive learning is mostly used in applications with structurally rich representations, e.g. in biology or medicine. A typical example is the carcinogenesis problem (see Section 7.2.2), where examples are chemical compounds. Each compound contains atoms, which are connected, have type information associated with them as well as results of chemical tests. However, for simpler learning problems, e.g. where examples are represented using a set of boolean features with no further background knowledge, there are usually more appropriate machine learning methods than inductive learning. Please note that,

in principle, different languages could be used for background knowledge, examples and hypothesis.

As a summary of the introduced notions, [Lavrac and Dzeroski, 1994] defines concept learning as follows:

Definition 2.20 (Concept Learning with Background Knowledge)

Given a set of training examples E and background knowledge B , find a hypothesis H , expressed in some concept description language L , such that H is complete and consistent [i.e. correct] with respect to the background knowledge B and the examples E . □

This concludes the general introduction of concept and inductive learning as well as its broader context in machine learning. Shortly, in Section 2.2.2 we will give specific formulations of the learning problems we are interested in within this thesis.

2.2.1 History, Tools, and Applications

As outlined above, one of the most prominent feature of inductive reasoning systems is the use of background knowledge. One of the earliest systems having this capability was INDUCE [Michalski, 1980], which used relational structures as background knowledge. The term Inductive Logic Programming was coined in the early 90s [Muggleton, 1991, Muggleton, 1992]. [Muggleton and Raedt, 1994] was a milestone in ILP research. Inductive Logic Programming turned into one of the most prominent machine learning research and application areas. Several books have been published in this area in the mid 90s [Lavrac and Dzeroski, 1994, Bergadano and Gunetti, 1995, Raedt, 1996, Nienhuys-Cheng and de Wolf, 1997]. While logic programs were the main formalism used in inductive reasoning, similar ideas were also used in relational databases [Blockeel and Raedt, 1996] as well as description logics [Cohen and Hirsh, 1994, Badea and Nienhuys-Cheng, 2000, Esposito et al., 2004], and combinations of DLs and rules [Lisi and Malerba, 2003, Lisi, 2005]. Over the last five years, Probabilistic ILP [Raedt et al., 2008] and Statistical Relation Learning [Raedt, 2005, Kersting, 2006] attracted interest. Those developments opened up several new application areas of inductive reasoning.¹²

After this brief overview, we give a list and short description of some existing learning systems. For understanding the system descriptions, familiarity with basic notions from logic programming and first order logics is required.

Progol ¹³ is an ILP system, which employs a covering approach like many other ILP systems. This means that it generates several clauses stepwise. If examples are covered by the current set of clauses, they are removed. This process continues until sufficiently many examples have been covered. For finding

¹²This thesis can be seen as a contribution to this trend, since ontology engineering is explored as application area and, more generally, OWL ontologies as background knowledge.

¹³<http://www.doc.ic.ac.uk/~shm/progol.html>

an appropriate clause, Progol selects a seed example and builds a most specific clause, also called bottom clause, using so called *inverse entailment*. This clause is generalised in the learning process. See [Muggleton, 1995, Muggleton, 1996a] for a more detailed explanation. Two variants of Progol are available: C-Progol (written in the C programming language) and P-Progol (written in Prolog).

Aleph¹⁴ (A Learning Engine for Proposing Hypotheses) is one of the most popular ILP systems and the successor of P-Progol. It is envisioned as a prototype for several algorithms and ideas and can actually simulate the behaviour of other learning systems.

Golem¹⁵ is based on the idea of relative least general generalisations (rlggs) introduced in Plotkin's PhD thesis [Plotkin, 1971]. It uses a covering approach (see the explanation of Progol above), where the idea of rlggs is employed for learning single clauses. The rlggs is a least general hypothesis relative to background knowledge with respect to given (positive) examples. Golem first computes the rlggs of randomly selected pairs of examples. The best result, i.e. the hypothesis with the best coverage of those random pairs' rlggs, is picked. In the next loop, the system computes the rlggs of the currently best hypothesis and randomly selected examples. Again, the best result is picked and the system continues until a further loop does not increase coverage of the hypothesis. Details are described in [Muggleton and Feng, 1990].

FOIL is a well-known ILP system, which inspired a lot of further research. The background knowledge is extensional, i.e. relational/ground tuples are used and the target language are function free Horn clauses. As other systems introduced before, it uses a coverage approach. The induction of a single clause starts with an empty clause body. The clause body is iteratively specialised by adding literals. The literals are chosen by placing variables to their appropriate argument places, which need to be specified before. Inequality between variables and the use of previously specified relevant constants are also allowed as literals. Literals have to conform to type restrictions declared by the user. Similar restrictions on literals are used in the more recent Progol and Aleph systems (see above) for controlling the language bias. To choose a literal amongst the possible candidates, FOIL uses the information gain heuristic, which is based on how much the literal helps in distinguishing between positive and negative examples. See [Quinlan and Cameron-Jones, 1993] for details.

MERLIN¹⁶ is a system, which supports positive only learning. It has the ability of learning meaningful logic programs from a single example using refutations

¹⁴<http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/aleph.html>

¹⁵<http://www.doc.ic.ac.uk/~shm/Software/golem/>

¹⁶<http://people.dsv.su.se/~henke/ML/MERLIN.html>

of SLD resolutions and finite-state automata [Boström, 1996].

LIVE¹⁷ (Learning in a three-Valued Environment) is a system, which learns two definitions given a learning problem: One definition for the positive examples and another one for the negative examples [Lamma et al., 1999]. As background knowledge, it uses Extended Logic Programs under the well-founded semantics extended with explicit negation (WFSX). It is based on Golem.

FOIDL¹⁸ learns first order decision lists and has been used successfully to learn the past tense of English verbs [Mooney and Califf, 1995].

The list is far from complete¹⁹, but highlights some of the principles of popular and/or interesting ILP systems. They vary in knowledge representation, solution search strategies, heuristics, and the types of learning problems to consider. Those learning systems have been applied in several scenarios. Some of those are outlined below:

Learning Drug structure activity rules is a major usage area of ILP systems. The aim is to understand the relationship between chemical structure and activity. Usually, the activity can only be determined using experiments. Conducting those experiments is, however, very expensive and time-consuming. ILP systems are used to learn rules relating the structure to an activity and can be used to find promising chemical structures, i.e. increase the success rate of pharmaceutical companies. Concrete problems include learning structure activity relationships for Alzheimer disease, inhibition of E. Coli Dihydrofolate Reductase, and suramin analogues. Due to the rich structure of background knowledge, ILP turned out to be very suitable (see [King et al., 1995] for details). The learned rules provided new insights and reductions in the number of chemical compounds to be tested.

Natural Language Processing provides several application areas for inductive reasoning: One example is learning lexical and grammatical knowledge for each language. This knowledge exists for the most common languages, but needs to be transferred to other languages. In [Kazakov, 1999, Mooney, 1997] natural language parsers have been learned inductively. Another application area is part of speech tagging. Inductive reasoning has been applied for several languages, including English [Cussens, 1996], in this area. ILP has also been used to extract relations from text [Horvath et al., 2009], which is useful for building knowledge bases from text corpora.

¹⁷<http://lia.deis.unibo.it/Software/live/>

¹⁸<http://www.cs.utexas.edu/users/ml/foidl.html>

¹⁹See <http://www-ai.ijs.si/~ilpnet2/systems/> for a list of more than 50 ILP systems, which have been created before 2003.

Robot Discovery is a research area, which aims to find out to which extent robots can learn about their environment by conducting experiments and collecting data. [Leban et al., 2008] describes experiments with a simulated and a real robot in a simple domain, where the robot learns the concepts of “movability” and “obstacle”.

Detection of Traffic Problems as an ILP application area has been explored in [Dzeroski et al., 1998, Dzeroski et al., 1998] with the goal of identifying problematic areas with respect to traffic jams and accidents. The AIMSUN simulator generates examples for car traffic around the city Barcelona. Different sensors with relevant information were recorded and the background knowledge contained the structure of the road network. The ILP tools Claudien and Tilde were applied to the problem setting. Those tools could identify most critical sections correctly and learn plausible rules.

It can be concluded that inductive reasoning is a rich and diverse research area with several applications, in particular those where rich structural knowledge is available. The short list is by no means exhaustive²⁰ and is only meant to give some examples of potential application scenarios of the algorithms developed in this thesis (see Chapter 7 for the applications of the DL-Learner framework). Despite the relative success of inductive reasoning in some fields, there are also some open issues.

One practical problem is to make results of ILP programs more readable and understandable for their users. For instance, the *Predictive Toxicology Challenge*²¹ was a competition between different ILP programs announced at the International Joint Conference on Artificial Intelligence in 1997. The challenge received several submissions until 1999. Later analysis of the results by experts in biology and chemistry [Benigni and Giuliani, 2003, Helma et al., 2001, Toivonen et al., 2003] showed that results were promising and confirmed existing or led to new knowledge. However, it was hard for domain experts to understand the obtained results, i.e. there is a gap between ILP researchers and users. Therefore, the readability of results is highly prioritized in the learning algorithms mentioned in this thesis. We also argue, although this matter is subjective, that OWL class expressions might be easier to understand than logic programs. See Section 7.6 for more discussion on those issues.

Another problem is scalability. At the most recent Inductive Logic Programming conference, it was argued that ILP programs do not scale well for large data sets [Watanabe and Muggleton, 2009]. One challenge is the size of the hypothesis space, in particular for expressive languages. However, there are (at least) two further reasons for scalability problems: Checking coverage of a hypothesis requires reasoning, which can be expensive depending on the used background language.

²⁰<http://www-ai.ijs.si/~ilpnet2/apps/index.html> and <http://www.doc.ic.ac.uk/~shm/applications.html> provide further pointers to applications.

²¹<http://www.comlab.ox.ac.uk/activities/machinelearning/PTE/>

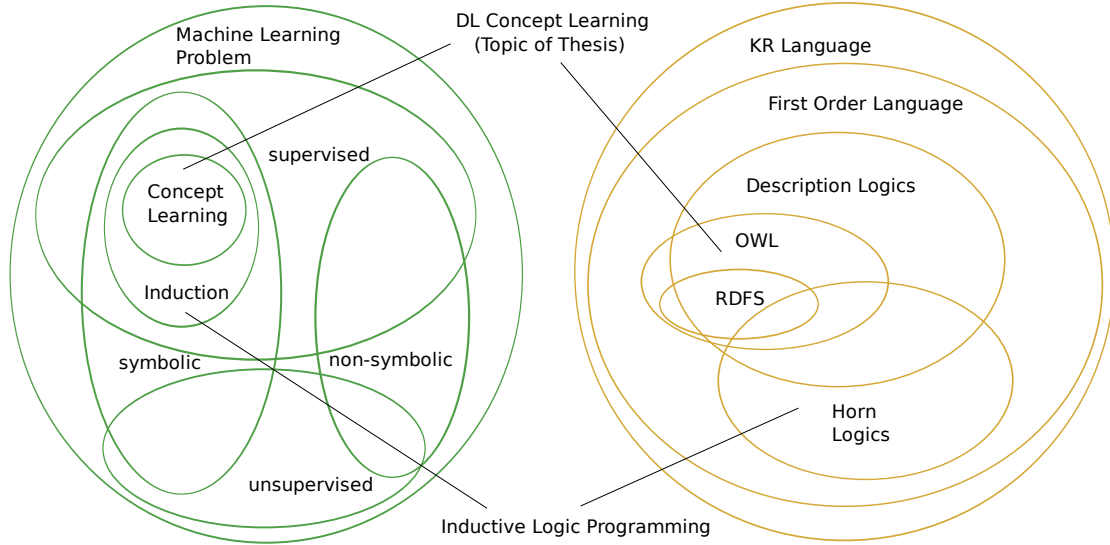


Figure 2.3: Overview of knowledge representation languages and machine learning problems to illustrate how this thesis integrates into existing research areas.

The other reason is that the runtime of many systems grows approximately linearly with the number of examples. Therefore, many (> 1000) examples can be a challenge. We devote Chapter 6 for the discussion of scalability issues and possible solutions.

2.2.2 Learning Problems in OWL/DLs

For learning in description logics we can give a more specific description of the setting previously introduced in Definition 2.20. The background knowledge is a knowledge base \mathcal{K} in some description language. Examples are objects contained in this knowledge base and hypothesis are complex concepts. Figure 2.3 illustrates how this integrates into existing research areas. The ellipses on the left represent machine learning problems and the ellipses on the right show knowledge representation languages ordered by expressivity of the language.

We distinguish three different variations of learning problems. In the first setting, we are given positive and negative examples explicitly and the goal is to find a concept, which covers the positives without covering the negatives:

Definition 2.21 (Learning From Examples in OWL/DLs)

Let a DL knowledge base \mathcal{K} and disjoint sets E^+ and E^- with $E \subseteq N_I$ where $E = E^+ \cup E^-$ be given. Learning from examples means to find a (complex) concept C such that $\mathcal{K} \models C(e)$ for all $e \in E^+$ and $\mathcal{K} \not\models C(e)$ for all $e \in E^-$. \square

In the second problem, we are only provided with positive examples. This occurs frequently in practice, since negative examples are not always available or

it is more natural to provide only positives. For instance, it is straightforward to provide the presidents of the United States, but less natural to provide typical examples of non-presidents. As an interesting side note, [Muggleton, 1996b] cites that studies have shown that children can learn in those settings. For instance, they are rarely informed of grammatical errors or do not pay attention to this information. Yet, they are still able to reduce the number of errors they are making over time. In a positive only learning setting, the goal is still to cover all positives. Additionally, a hypothesis should not cover additional individuals in the knowledge base. Technically, this could be realised by using $E^- = N_I \setminus E^+$, but adapted techniques can be more promising.

Definition 2.22 (Positive Only Learning in OWL/DLs)

Let a DL knowledge base \mathcal{K} and a set E^+ with $E^+ \subseteq N_I$ be given. The positive only learning problem is to find a (complex) concept C such that $R_{\mathcal{K}}(C) = R_{\mathcal{K}}(E^+)$. \square

In the third learning problem we consider, the goal is to learn a description of a named concept A in a knowledge base. The main difference compared to the previous problems is that existing knowledge about A in the background knowledge can be used if available, e.g. its position in the subsumption hierarchy.

Definition 2.23 (Class/Concept Learning in OWL/DLs)

Let a concept name $A \in N_C$ and a knowledge base \mathcal{K} be given. The *class/concept learning problem* is to find a concept C such that $R_{\mathcal{K}}(C) = R_{\mathcal{K}}(A)$. \square

The learned concept C is a description of (the instances of) A . Such a concept is a candidate for adding an axiom of the form $A \equiv C$ or $A \sqsubseteq C$ to the knowledge base \mathcal{K} . We give a brief example of a simple class learning problem.

Example 2.24 (Concept Learning Example)

As background knowledge, consider the SWORE [Riechert et al., 2007b] knowledge base, which is concerned with requirements engineering. The central goal of requirements engineering is to collect requirements for software or hardware. SWORE contains the concept `CustomerRequirement`. A machine learning algorithm could suggest that this concept is equivalent to:

$$\begin{aligned} \text{Requirement} &\sqcap \exists \text{createdBy.Customer} \\ \text{AbstractRequirement} &\sqcap \exists \text{createdBy.Customer} \end{aligned}$$

Both suggestions could be plausible and equally accurate in this case. It is the task of the knowledge engineer to decide which one is more appropriate. \square

Some remarks about the learning problems are in order. First of all, there are slightly different formulations of those problems in the literature. A possible different setting is that negative examples have to be instances of $\neg C$, i.e. $\mathcal{K}' \models \neg C(e)$ for all $e \in E^-$. Due to the open world assumption in description logics there is a

difference between facts logically following from a knowledge base and the negation of a fact following from the knowledge base – unlike in logic programming with default negation. Both ways of formulating the learning problem are meaningful. However, it should be noted that $\mathcal{K} \models \neg C(a)$ can only be inferred if there is no model \mathcal{I} , where $a^{\mathcal{I}} \in C^{\mathcal{I}}$, i.e. there is no possible world where a belongs to C . Many knowledge bases do not contain sufficiently many restrictions to allow such inferences to be drawn. For this reason, it is more common to require that negative examples are not instance of the hypothesis, as we have done in the definitions above. It is, however, straightforward to apply the learning algorithms in this thesis to both settings.

It should also be noted that in most cases, an algorithm will not find a correct solution to the learning problem, but rather an approximation. This is natural, since a knowledge base may contain false class assignments or some objects in the knowledge base are described at different levels of detail. For instance, in Example 2.24, certain requirements created by customers might not be typed as such in the knowledge base. However, if most of the other requirements are related to countries via a role `createdBy` to instances of `Customer`, then the learning algorithm may still suggest the shown complex concepts, since they describe the majority of customer requirements in the knowledge base well. If the knowledge engineer agrees with such a definition, then a tool can assist him in completing missing information about some customer requirements.

By Occam’s razor [Blumer et al., 1990, Domingos, 1998] simple solutions of the learning problem are to be preferred over more complex ones, because they are more readable. This is even more important in the ontology engineering context, where it is essential to suggest simple concepts to the knowledge engineer. We measure simplicity as the *length* of a concept as defined in Section 2.1.3.

Also note that the definitions of the learning problems itself do enforce coverage, but not prediction, i.e. correct classification of objects which are added to the knowledge base in the future. Concepts with high coverage and poor prediction are said to *overfit* the data. Learning algorithms have to take care to avoid overfitting, e.g. by biasing towards short concepts.

2.2.3 Refinement Operators in OWL/DLs

The goal of learning is to find a correct concept with respect to the examples. This can be seen as a search process in the space of concepts, which is illustrated in Figure 2.4. A concept generator provides new hypothesis to be tested, which are evaluated using a heuristic measure. Apart from other criteria, this heuristic usually uses the provided examples and a DL reasoner to perform a coverage test of the hypothesis. Each evaluation assigns a score to the given hypothesis, which can be taken into account by the concept generator. This is described in more detail in Section 5. An intelligent way to suggest new hypothesis is a key problem in defining a learning algorithm.

One idea to solve this problem is to refine promising generated hypothesis. A

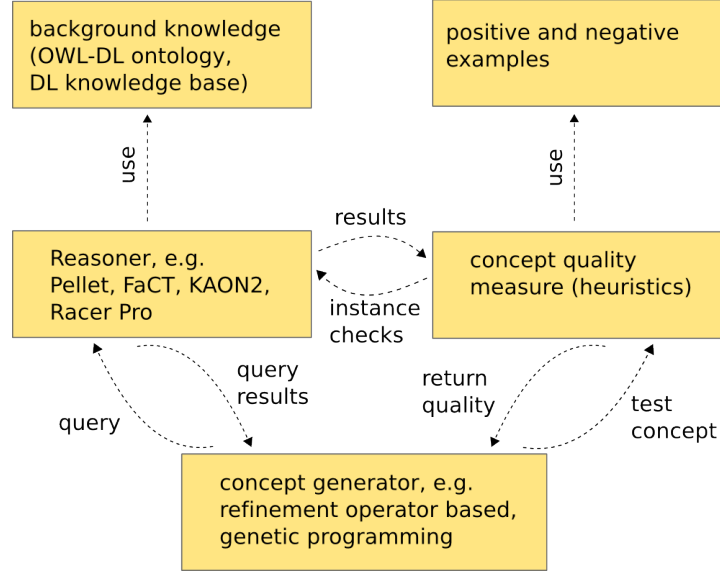


Figure 2.4: Generate and test approach used in DL-Learner.

natural way to structure the search space is to impose an ordering and use operators to traverse it. This approach is well-known in Inductive Logic Programming, where refinement operators are widely used to find hypotheses. Intuitively, downward (upward) refinement operators construct specialisations (generalisations) of hypotheses.

Definition 2.25 (Refinement Operator)

A *quasi-ordering* is a reflexive and transitive relation. In a quasi-ordered space (S, \preceq) a *downward (upward) refinement operator* ρ is a mapping from S to 2^S , such that for any $C \in S$ we have that $C' \in \rho(C)$ implies $C' \preceq C$ ($C \preceq C'$). C' is called a *specialisation (generalisation)* of C . \square

This idea can be used for searching in the space of concepts. As ordering we can use subsumption. (Note that the subsumption relation \sqsubseteq is a quasi-ordering.) If a concept C subsumes a concept D ($D \sqsubseteq C$), then C will cover all examples which are covered by D . This makes subsumption a suitable order for searching in concepts. We analyse refinement operators for concepts with respect to subsumption and a description language \mathcal{L} , and in the sequel we will call such operators \mathcal{L} *refinement operators*. We also introduce the commonly used notions of refinement chains as well as downward and upward covers.

Definition 2.26 (\mathcal{L} Refinement Operator)

Let \mathcal{L} be a description language. A refinement operator in the quasi-ordered space $(\mathcal{L}, \sqsubseteq)$ is called an \mathcal{L} *refinement operator*. \square

Definition 2.27 (Refinement Chain)

A *refinement chain* of an \mathcal{L} refinement operator ρ of length n from a concept C to a concept D is a finite sequence C_0, C_1, \dots, C_n of concepts, such that $C = C_0, C_1 \in$

$\rho(C_0), C_2 \in \rho(C_1), \dots, C_n \in \rho(C_{n-1}), D = C_n$. This refinement chain *goes through* E iff there is an i ($1 \leq i \leq n$) such that $E = C_i$. We say that D can be reached from C by ρ if there exists a refinement chain from C to D . $\rho^*(C)$ denotes the set of all concepts, which can be reached from C by ρ . $\rho^m(C)$ denotes the set of all concepts, which can be reached from C by a refinement chain of ρ of length m . \square

Definition 2.28 (Downward and Upward Cover)

A concept C is a *downward cover* of a concept D iff $C \sqsubset D$ and there does not exist a concept E with $C \sqsubset E \sqsubset D$. A concept C is an *upward cover* of a concept D iff $D \sqsubset C$ and there does not exist a concept E with $D \sqsubset E \sqsubset C$. \square

Instead of $D \in \rho(C)$, we will often write $C \rightsquigarrow_\rho D$. If the used operator is clear from the context, it is usually omitted, i.e. we write $C \rightsquigarrow D$.

We introduce the notion of weak equality of concepts, which is similar to syntactic equality of concepts, but takes commutativity into account, i.e. the order of elements in conjunctions and disjunctions is not important. We say that the concepts C and D are *weakly (syntactically) equal*, denoted by $C \simeq D$ iff they are equal up to permutation of arguments of conjunction and disjunction. Two sets S_1 and S_2 of concepts are weakly equal iff for any $C_1 \in S_1$ there is a $C'_1 \in S_2$ such that $C_1 \simeq C'_1$ and vice versa. Weak equality of concepts is coarser than syntactic equality and finer than equivalence (viewing the equivalence, equality, and weak equality of concepts as equivalence classes).

Refinement operators can have certain properties, which can be used to evaluate their usefulness for learning hypotheses.

Definition 2.29 (Properties of DL Refinement Operators)

An \mathcal{L} refinement operator ρ is called

- *(locally) finite* iff $\rho(C)$ is finite for any concept C .
- *redundant* iff there exists a refinement chain from a concept C to a concept D , which does not go through some concept E and a refinement chain from C to a concept weakly equal²² to D , which does go through E .
- *proper* iff for all concepts C and D , $D \in \rho(C)$ implies $C \neq D$.
- *ideal* iff it is finite, complete (see below), and proper.

An \mathcal{L} downward refinement operator ρ is called

- *complete* iff for all concepts C, D with $C \sqsubset D$ we can reach a concept E with $E \equiv C$ from D by ρ .

²²We use weak equality instead of syntactic equality, because only avoiding syntactic equality, while still being able to reach many weakly equal concepts from a given concept, can still waste significant computational resources. All results in Chapter 3 also hold if we consider syntactic equality, see Remark 3.18.

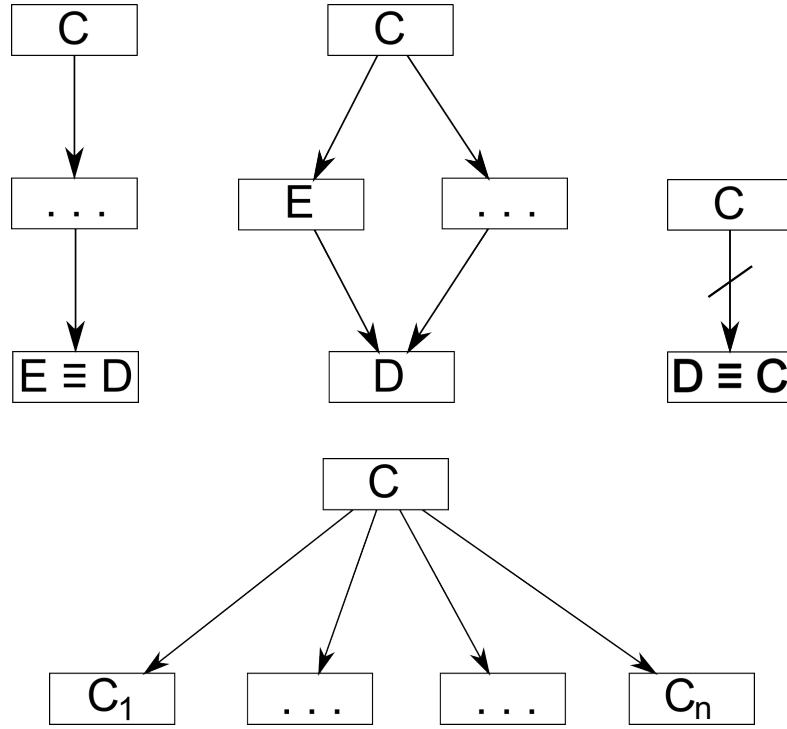


Figure 2.5: Illustration of some refinement operator properties (left to right): completeness, redundancy, properness, and finiteness (bottom).

- *weakly complete* iff for all concepts $C \sqsubseteq \top$ we can reach a concept E with $E \equiv C$ from \top by ρ .
- *minimal* iff for all C , $\rho(C)$ contains only downward covers and all its elements are incomparable with respect to \sqsubseteq .

The corresponding notions for upward refinement operators are defined dually. \square

3 Theoretical Foundations of Refinement Operators

In this chapter, we analyse the properties of refinement operators in description logics. The need for such an analysis was already expressed in [Fanizzi et al., 2004, Esposito et al., 2004]. In particular, we are interested in finding out which desired properties can be combined in a refinement operator and which properties are impossible to combine. This is interesting for two reasons: Firstly, this gives us a good impression of how hard (or easy) it is to learn concepts. Secondly, this can also serve as a practical guide for designing refinement operators. Knowing the theoretical limits allows the designer of a refinement operator to focus on achieving the best possible properties. We will indeed follow this approach in Chapter 4, where we will define – and later evaluate – concrete operators based on our theoretical investigations.

Refinement operators for description logics have been constructed for \mathcal{ALER} in [Badea and Nienhuys-Cheng, 2000], for \mathcal{ALN} in [Fanizzi et al., 2004], and for \mathcal{ALC} in [Esposito et al., 2004, Iannone and Palmisano, 2005, Iannone et al., 2007]. In particular, [Badea and Nienhuys-Cheng, 2000] also showed some properties of \mathcal{ALER} refinement operators. However, a full theoretical treatment of their properties has not been done to the best of our knowledge (not even for a specific language). Therefore, all propositions in this section are new unless explicitly mentioned otherwise. The main result of the chapter is Theorem 3.16, which provides a full analysis of combinations of refinement operator properties. In this chapter, we assume that each language we analyse contains or can express top (\top) and bottom (\perp). We also assume finite sets N_C and N_R of concept and role names in this PhD thesis containing the concept and role names occurring in the considered knowledge base.

3.1 The Role of Minimality

As a first property we will briefly analyse minimality of \mathcal{L} refinement operators, in particular the existence of upward and downward covers in \mathcal{ALC} . It is not immediately obvious that e.g. downward covers exist in \mathcal{ALC} , because it could be the case that for any concept C and D with $C \sqsubset D$ one can always construct a concept E with $C \sqsubset E \sqsubset D$. However, the next proposition shows that downward covers do exist.

Proposition 3.1 (Existence of Covers in \mathcal{ALC})

Downward (upward) covers of \top (\perp) exist in \mathcal{ALC} .

PROOF Let $N_R = \{r\}$ and $N_C = \{A\}$.

Assume, we have an interpretation \mathcal{I} and an object a such that $a^{\mathcal{I}} \notin A^{\mathcal{I}}$ and there is no b with $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ (*). We define a set S as follows:

- $\top \in S$
- $\neg A \in S$
- for all concepts C , $\forall r.C \in S$
- if $C_1 \in S$ and $C_2 \in S$, then $C_1 \sqcap C_2 \in S$
- if $C_1 \in S$ or $C_2 \in S$, then $C_1 \sqcup C_2 \in S$

By structural induction on \mathcal{ALC} concept constructors (see Definition 2.5 and Table 2.1), it is not hard to show that $C \in S$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all C in negation normal form.

Using this observation as prerequisite, we show that $C = \exists r.\top \sqcup A$ is a downward cover of \top . By contradiction, we assume that there is a concept D with $C \sqsubset D \sqsubset \top$ in negation normal form.

Since $C \sqsubset D$, there are \mathcal{I}_1 and a_1 such that $a_1^{\mathcal{I}_1} \notin C^{\mathcal{I}_1}$ and $a_1^{\mathcal{I}_1} \in D^{\mathcal{I}_1}$. Analogously, due to $D \sqsubset \top$, there are \mathcal{I}_2 and a_2 such that $a_2^{\mathcal{I}_2} \notin D^{\mathcal{I}_2}$. By $C \sqsubset D$, this implies $a_2^{\mathcal{I}_2} \notin C^{\mathcal{I}_2}$. In summary (**):

$$\begin{array}{ll} I_1: & a_1^{\mathcal{I}_1} \notin C^{\mathcal{I}_1} \quad a_1^{\mathcal{I}_1} \in D^{\mathcal{I}_1} \\ I_2: & a_2^{\mathcal{I}_2} \notin C^{\mathcal{I}_2} \quad a_2^{\mathcal{I}_2} \notin D^{\mathcal{I}_2} \end{array}$$

We can deduce:

$$\begin{aligned} & a_1^{\mathcal{I}_1} \notin C^{\mathcal{I}_1} \\ \iff & a_1^{\mathcal{I}_1} \notin (A \sqcup \exists r.\top)^{\mathcal{I}_1} \\ \iff & a_1^{\mathcal{I}_1} \notin A^{\mathcal{I}_1} \text{ and } a_1^{\mathcal{I}_1} \notin (\exists r.\top)^{\mathcal{I}_1} \\ \iff & a_1^{\mathcal{I}_1} \notin A^{\mathcal{I}_1} \text{ and there is no } b \text{ with } (a^{\mathcal{I}_1}, b^{\mathcal{I}_1}) \in r^{\mathcal{I}_1} \end{aligned}$$

The same can be done for a_2 and \mathcal{I}_2 . In both cases (*) is satisfied.

If $D \in S$, then $a_1^{\mathcal{I}_1} \in D^{\mathcal{I}_1}$ and $a_2^{\mathcal{I}_2} \in D^{\mathcal{I}_2}$. Otherwise if $D \notin S$, then $a_1^{\mathcal{I}_1} \notin D^{\mathcal{I}_1}$ and $a_2^{\mathcal{I}_2} \notin D^{\mathcal{I}_2}$. Both cases contradict (**).

Upward covers can be handled analogously, i.e. $\forall r.\perp \sqcap A$ is an upward cover of \perp . ■

In the proof we have shown that $\exists r. \top \sqcup A$ is a downward cover of \top for the case that there is only one role and one concept name. The idea can be extended to situations with more than one role and concept name. In this case we obtain the following concept as a downward cover of \top (we do not prove this explicitly, because we do not use this result later on):

$$\bigsqcup_{r \in N_R} \exists r. \top \sqcup \bigsqcup_{A \in N_C} A \quad (3.1)$$

The result shows that non-trivial minimal operators, i.e. operators which do not map every concept to the empty set, can be constructed. However, it is also apparent that in practice Concept 3.1 is long for realistic knowledge bases.

It should be noted that minimality of refinement steps is not a directly desired goal in general. Minimal operators are in some languages more likely to lead to overfitting, because they may not produce sufficient generalisation/specialisation leaps, e.g. the cover above provides almost no specialisation compared to \top . This problem is particularly significant in languages which are closed under boolean operations, i.e. in \mathcal{ALC} and more expressive languages.

Indeed, the following result suggests that – unlike for logic programs – minimality may not play a central role for DL refinement operators, as it is incompatible with weak completeness. In [Badea and Nienhuys-Cheng, 2000], a weaker result was already claimed to hold, but not proven. We formulate our result for the description logic \mathcal{AL} (see page 26 for its definition). A corresponding result for other description logics than \mathcal{AL} has not been shown yet, but the non-existence of such operators even for weak description logics suggests that similar problems arise for more expressive ones.

Proposition 3.2 (minimality and weak completeness)

There exists no minimal and weakly complete \mathcal{AL} downward refinement operator.

PROOF Let $N_R = \{r\}$ and $N_C = \emptyset$. In the following, let $\text{rd}(C)$ denote the role depth of a concept C .

For a proof by contradiction, we assume a minimal and weakly complete \mathcal{AL} downward refinement operator ρ . This implies that there is a refinement step $C \rightsquigarrow_\rho D$, such that C does not have a subconcept equivalent to \perp and D does not have a subconcept equivalent to \perp . Otherwise no concept equivalent to \perp would be reachable from \top . We prove the proposition by first simplifying C and D and then defining a concept E with $D \sqsubset E \sqsubset C$.

We next cast C and D into a normal form. For this, we replace the subconcepts in D equivalent to \perp by \perp . Furthermore, we apply the following equivalence

preserving rewrite rules exhaustively to C and D :

$$\begin{aligned} C \sqcap \perp &\rightarrow \perp & \text{and} & \quad \perp \sqcap C \rightarrow \perp \\ C \sqcap \top &\rightarrow C & \text{and} & \quad \top \sqcap C \rightarrow C \\ \forall r. \top &\rightarrow \top \\ \forall r. (C_1 \sqcap C_2) &\rightarrow \forall r. C_1 \sqcap \forall r. C_2 \end{aligned}$$

We pick a normal form obtained this way (note that applying the rules to an arbitrary concept can lead to different results depending on the order of application) and call the resulting concepts $C' (\equiv C)$ and $D' (\equiv D)$.

Due to the syntax of \mathcal{AL} , $N_C = \emptyset$, and the rewriting rules, we have that C' is either \top or of the following form, where the s_i , for $i \in \{1, \dots, b\}$, are non-negative integers:

$$C' = \bigcap_{i=1}^b \forall r^{s_i}. \exists r. \top \quad (3.2)$$

We define a new concept E :

$$E = C' \sqcap \forall r^n. \exists r. \top \quad \text{with } n = \max(\text{rd}(C), \text{rd}(D)) + 1$$

We complete the proof by showing that $D' \sqsubset E \sqsubset C'$, which contradicts the minimality of ρ .

1. To show $E \sqsubset C'$, first note that $E = C' \sqcap \forall r^n. \exists r. \top \sqsubseteq C'$ is obvious, so it remains to show that E and C' are not equivalent. To do this, we define \mathcal{I} and a such that $a^{\mathcal{I}} \in C'$ and $a^{\mathcal{I}} \notin E$.

Let \mathcal{I} be defined by $r^{\mathcal{I}} = \{(a_i, a_{i+1}) \mid a_0 = a, 0 \leq i < n\}$, which we can depict as

$$a = a_0 \xrightarrow{r} a_1 \xrightarrow{r} \dots \xrightarrow{r} a_n.$$

- Indeed $a^{\mathcal{I}} \in C'$ holds: For $C' = \top$ this holds trivially. If $C' \neq \top$, then C' is a conjunction of concepts which are of the form $\forall r^m. \exists r. \top$ with $m < n$, i.e. we have $a^{\mathcal{I}} \in (\forall r^m. \exists r. \top)^{\mathcal{I}}$ for any $m < n$. Note that the statement $\forall r^m. \exists r. \top$ informally means that objects reachable via a path of length m along r need to have a successor. Hence $a^{\mathcal{I}} \in C'$ holds due to the form we have established in (3.2) for C' .
 - $a^{\mathcal{I}} \notin E$ holds because a_n does not have an r -filler.
2. To show $D' \sqsubset E$, first note that D' contains a \perp symbol. This means that D' is of the form $D' = D'_1 \sqcap \dots \sqcap D'_p$ ($p \geq 1$) where there exists some $j \in \{1, \dots, p\}$ s.t. D'_j is of the form $\forall r^t. \perp$, for some $t \geq 0$. Now define $F = C' \sqcap D'_j$. Note that¹ $D' \equiv C' \sqcap D' \sqsubseteq C' \sqcap D'_j = F$.

¹We have $D' \equiv C' \sqcap D'$, because $D \equiv D'$ is a downward refinement of $C \equiv C'$, i.e. $D' \sqsubseteq C'$. $C' \sqcap D' \sqsubseteq C' \sqcap D'_j$ holds because of $D' \sqsubseteq D'_j$.

We first verify $D' \sqsubseteq E$. Note that for all $k, l \geq 0$ we have

$$\forall r^k.\perp \sqsubseteq \forall r^k.\forall r^l.\exists r.\top,$$

and since $t < n$ we obtain

$$D' \sqsubseteq F = C' \sqcap \forall r^t.\perp \sqsubseteq C' \sqcap \forall r^n.\exists r.\top = E$$

It remains to show that $D' \not\sqsubseteq E$. We do this by proving $F \not\sqsubseteq E$ using the interpretation \mathcal{I} with $r^{\mathcal{I}} = \{(a, a)\}$.

We have $a^{\mathcal{I}} \in (\forall r^z.\exists r.\top)^{\mathcal{I}}$ for all $z \geq 0$. Hence, we have $a^{\mathcal{I}} \in C'^{\mathcal{I}}$ and also $a^{\mathcal{I}} \in E^{\mathcal{I}}$.

We have $a^{\mathcal{I}} \notin F^{\mathcal{I}}$, because $a^{\mathcal{I}} \notin (\forall r^t.\perp)^{\mathcal{I}}$ – note that $\forall r^t.\perp$ states that objects reachable via a path of length $t - 1$ must not have a successor.

Hence, $a^{\mathcal{I}} \in E^{\mathcal{I}}$ and $a^{\mathcal{I}} \notin F^{\mathcal{I}}$, which proves $F \not\sqsubseteq E$.

We have shown that there is a refinement step $C \rightsquigarrow_{\rho} D$ such that a concept E exists, which is more special than C and more general than D , i.e. ρ is not minimal. ■

Please note that although we looked at the special case of $N_R = \{r\}$ and $N_C = \emptyset$ in the proof, similar arguments could be used for arbitrary N_R and N_C .

This negative result for a very simple description logic indicates that minimality and (weak) completeness cannot be combined for many other expressive description logics. However, (weak) completeness is an important property, since a learning algorithm using an incomplete operator may miss potential solutions of a learning problem. Therefore, we focus on combinations of other properties than minimality in the remainder of the chapter.

3.2 Combinations of Completeness, Properness, Finiteness, Redundancy

In the sequel, we analyse desired properties of \mathcal{L} refinement operators: completeness, properness, finiteness, and non-redundancy. As we have just shown that minimality is unlikely to play a central role when investigating expressive DLs, we omit it from further investigations. We show several positive and negative results, which together yield a full analysis of these properties in Theorem 3.16.

Proposition 3.3 (Complete and Finite Refinement Operators)

Let \mathcal{L} be a description language which allows conjunction. Then there exists a complete and finite \mathcal{L} refinement operator.

PROOF Consider the downward refinement operator ρ defined by

$$\rho(C) = \{C \sqcap \top\} \cup \{D \mid |D| \leq (\text{number of } \top \text{ occurrences in } C) \text{ and } D \sqsubset C\},$$

where $|D|$ stands for the number of symbols in D . The operator can do one of two things:

- add a \top symbol
- generate the set of all concepts up to a certain length, which are subsumed by C

The operator is finite, because the set of all concepts up to a given length is finite (and the singleton set $\{C \sqcap \top\}$ is finite).

The operator is complete, because given a concept C we can reach an arbitrary concept D with $D \sqsubset C$. This is obvious, because we only need to add \top -symbols until there are $|D|$ occurrences of \top . Within the next step we can then be sure to reach D .

For upward refinement operators we can use an analogous operator ϕ , which is also complete and finite:

$$\phi(C) = \{C \sqcap \top\} \cup \{D \mid |D| \leq (\text{number of } \top \text{ occurrences in } C) \text{ and } C \sqsubset D\}$$

■

Remark 3.4 (Complete and Finite Refinement Operators)

It has been claimed in [Badea and Nienhuys-Cheng, 2000] that complete and finite $\mathcal{AL}\mathcal{ER}$ refinement operators do not exist. However, this is refuted by Proposition 3.3. □

Of course, it is obvious that the operator used to prove Proposition 3.3 is not useful in practice, since it merely generates concepts without paying attention to an efficient traversal of the search space. However, here we are interested in theoretical limits of refinement operators. It is indeed difficult to design a good complete and finite refinement operator. The reason is that finiteness can only be achieved by using non-proper refinement steps (for our operator this was done by adding \top symbols). We will now prove this, i.e. show that it is impossible to define a complete, finite, and proper refinement operator. Such operators are known as ideal and their non-existence indicates that learning concepts in sufficiently expressive description logics is hard.

Lemma 3.5 (Non-Ideality Helper Lemma 1)

Let $C = \neg\exists r^n.\top \sqcup \exists r^{n+1}.\top$. There is no concept D with $C \sqsubseteq D \sqsubset \top$ and role depth smaller n in \mathcal{ALC} or \mathcal{ALCQ} .

PROOF Note that C is not equivalent to \top . For the interpretation \mathcal{I} with $r^{\mathcal{I}} = \{(a_i, a_{i+1}) \mid 0 \leq i < n\}$, illustrated by $a_0 \xrightarrow{r^{\mathcal{I}}} a_1 \xrightarrow{r^{\mathcal{I}}} \dots \xrightarrow{r^{\mathcal{I}}} a_n$, we have $a_0^{\mathcal{I}} \notin C^{\mathcal{I}}$.

By contradiction, we assume that such a concept D exists.

We can view an interpretation \mathcal{I} as a directed graph with respect to r in a straightforward way: The set of nodes is $\{b^{\mathcal{I}} \mid b \in N_I\}$ and the edges are $\{(b, c) \mid (b, c) \in r^{\mathcal{I}}\}$. We define $\text{lp}_{\mathcal{I},a}$ as the length of the longest paths in the graph of \mathcal{I} starting from $a^{\mathcal{I}}$ ($a \in N_I$). If such a path is infinite, we set $\text{lp}_{\mathcal{I},a} = \infty$.

Let \mathcal{I} be an arbitrary interpretation and a an arbitrary object. If $\text{lp}_{\mathcal{I},a} > n$, we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$ because of $a^{\mathcal{I}} \in (\exists r^{n+1}. \top)^{\mathcal{I}}$ and $\exists r^{n+1}. \top \sqsubseteq C$. If $\text{lp}_{\mathcal{I},a} < n$, we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$, because of $a^{\mathcal{I}} \in (\neg \exists r^n. \top)^{\mathcal{I}}$ and $\neg \exists r^n. \top \sqsubseteq C$. If $\text{lp}_{\mathcal{I},a} = n$, we have $a^{\mathcal{I}} \notin C^{\mathcal{I}}$ because of $a^{\mathcal{I}} \notin (\exists r^{n+1}. \top)^{\mathcal{I}}$ and $a^{\mathcal{I}} \notin (\neg \exists r^n. \top)^{\mathcal{I}}$. So we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$ iff $\text{lp}_{\mathcal{I},a} \neq n$.

Due to $D \not\sqsubseteq \top$, we know that $\neg D$ is satisfiable. \mathcal{ALC} , \mathcal{ALCQ} , and other description logics have the tree model property, i.e. any satisfiable concept has a model, where the graph we defined is tree shaped [Baader et al., 2007a]. In particular, this means that if there is a path in the model graph between two arbitrary objects, then this path is unique. So without loss of generality, we can assume that the graphs of the considered models are tree shaped. Let \mathcal{I} be a tree shaped model of $\neg D$ and a be an object such that $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$. Note that because of $C \sqsubseteq D$, we know $a^{\mathcal{I}} \notin C^{\mathcal{I}}$.

We know that $\text{lp}_{\mathcal{I},a} = n$ in \mathcal{I} , otherwise $a^{\mathcal{I}} \in C^{\mathcal{I}}$ as we have shown above. Let $a^{\mathcal{I}} = a_0 \xrightarrow{r^{\mathcal{I}}} a_1 \xrightarrow{r^{\mathcal{I}}} \dots \xrightarrow{r^{\mathcal{I}}} a_n$ be one of the longest paths in the graph of \mathcal{I} starting in a . We create a new interpretation \mathcal{I}' from \mathcal{I} by adding a new object a_{m+1} and changing $r^{\mathcal{I}}$ to $r^{\mathcal{I}'} = r^{\mathcal{I}} \cup \{(a_m, a_{m+1})\}$.

The following concept constructors involving roles are included in the mentioned languages: $\forall r$, $\exists r$, $\leq m r$, $\geq m r$. Semantically, all of those refer to role fillers (see Table 2.1), i.e. to neighbours in the interpretation graph defined above. Since the role depth of D is smaller than n and there is exactly one path in the graph of \mathcal{I}' from $a^{\mathcal{I}}$ to a_{m+1} , we can deduce $a^{\mathcal{I}'} \in (\neg D)^{\mathcal{I}'}$ from $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$ (the addition of a_{m+1} does not influence whether $a^{\mathcal{I}'} \in (\neg D)^{\mathcal{I}'}$ or not). However, \mathcal{I}' has $\text{lp}_{\mathcal{I}',a} = n+1$, because we added a_{m+1} . Thus, $a^{\mathcal{I}} \in C^{\mathcal{I}}$ as shown above and hence $a^{\mathcal{I}} \in D^{\mathcal{I}}$, which contradicts $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$. ■

Lemma 3.6 (Non-Ideality Helper Lemma 2)

Let

$$\begin{aligned} C &= \neg \exists r^n. \top \sqcup \exists r^{n+1}. \top \\ &\sqcup \geq 2 r. \top \sqcup \exists r. \geq 2 r. \top \sqcup \dots \sqcup \exists r^{n-1}. \geq 2 r. \top \\ &\sqcup \exists r^{-1}. \top \sqcup \exists r. \geq 2 r^{-1}. \top \sqcup \exists r^2. \geq 2 r^{-1}. \top \sqcup \dots \sqcup \exists r^n. \geq 2 r^{-1}. \top \end{aligned}$$

There is no concept D with $C \sqsubseteq D \sqsubset \top$ and role depth smaller n in \mathcal{SHOIN} or \mathcal{SROIQ} .

PROOF We use the same notions as in Lemma 3.5. C is not equivalent to \top by the same example as in the previous lemma. Again, we assume by contradiction that such a concept D exists. As before, there are a and \mathcal{I} with $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$ (and thus $a^{\mathcal{I}} \notin C^{\mathcal{I}}$). The difference is that \mathcal{SROIQ} and \mathcal{SHOIN} do not have the tree model property.

Since $\neg \exists r^n. \top \sqcup \exists r^{n+1}. \top \sqsubseteq C$, we know that $\text{lp}_{\mathcal{I},a} = n$ by the observations made in Lemma 3.5. Let $a^{\mathcal{I}} = a_0 \xrightarrow{r^{\mathcal{I}}} a_1 \xrightarrow{r^{\mathcal{I}}} \dots \xrightarrow{r^{\mathcal{I}}} a_n$ be the longest path in the graph of \mathcal{I} starting in a . Using $a^{\mathcal{I}} \notin C^{\mathcal{I}}$, we can make some observations concerning the graph of \mathcal{I} :

- a_0 has no incoming edge (due to $\exists r^{-1}. \top$)
- a_1 to a_n have exactly one incoming edge (due to $\exists r^i. \geq 2 r^{-1}. \top$ for $1 \leq i \leq n$)
- a_n does not have an outgoing edge (due to $\text{lp}_{\mathcal{I},a} = n$)
- a_0 to a_{n-1} have exactly one outgoing edge (due to $\exists r^i. \geq 2 r. \top$ for $0 \leq i \leq n-1$)

This means that there is a unique longest path, which forms a connected component of the interpretation graph. The concept constructors involving roles in \mathcal{SHOIN} and \mathcal{SROIQ} are $\forall r, \forall r^{-1}, \exists r, \exists r^{-1}, \leq m r, \leq m r^{-1}, \geq m r, \geq m r^{-1}, \exists r.\text{Self}$, all of which refer to direct neighbours (via incoming and outgoing edges) of an object in the graph.

Again, we can construct an interpretation \mathcal{I}' by adding a new object a_{m+1} to the longest path. Since the role depth of D is smaller than n , we can deduce $a^{\mathcal{I}'} \in (\neg D)^{\mathcal{I}'}$ from $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$. However, \mathcal{I}' has $\text{lp}_{\mathcal{I},a} = n+1$, because we added a_{m+1} . Thus, $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $a^{\mathcal{I}} \in D^{\mathcal{I}}$, which contradicts $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$. ■

Proposition 3.7 (Ideal Refinement Operators)

There does not exist any ideal \mathcal{ALC} , \mathcal{ALCQ} , \mathcal{SHOIN} , or \mathcal{SROIQ} downward refinement operator.

PROOF By contradiction, we assume that there exists an ideal downward refinement operator ρ . We further assume that there is a role $r \in N_R$. Let $\rho(\top) = T = \{C_1, \dots, C_n\}$ be the set of refinements of the \top concept. Due to finiteness of ρ , T has to be finite. Let n be a natural number larger than the maximum of the role depths of concepts in T . Due to Lemma 3.5 and 3.6, we know that there does not exist a more general concept than C (where C is as defined in the corresponding lemma depending on the language we consider), which is not equal to \top , with a role depth smaller than n . We have also shown that $C \not\equiv \top$.

Hence C_1, \dots, C_n do not subsume C (the properness of ρ implies that C_1, \dots, C_n are not equivalent to \top), so C cannot be reached from any of these concepts by applying ρ . Thus, C cannot be reached from \top and ρ is incomplete. ■

We limited the proof to a set of interesting languages. However, it generalizes to other languages with only minor adaptations.

Without the finiteness restriction, completeness and properness can be combined:

Proposition 3.8 (Complete and Proper Refinement Operators)

Let \mathcal{L} be a description language. Then there exists a complete and proper \mathcal{L} refinement operator.

PROOF To prove this, we can use $\rho(C) = \{D \mid D \sqsubset C\}$ as downward refinement operator, which is obviously complete and proper. For upward refinement we can analogously consider $\rho(C) = \{D \mid C \sqsubset D\}$. ■

We have shown that the combination of completeness and properness is possible. Propositions 3.3, 3.7, and 3.8 state that for complete refinement operators, which are usually desirable, one has to sacrifice properness or finiteness. We will now look at non-redundancy. We show two results: 1.) Completeness and non-redundancy can be combined theoretically (Proposition 3.9) and 2.) it is very unlikely that useful operators with this combination exist (Proposition 3.10). We argue that for practical purposes, completeness and non-redundancy cannot be combined.

Proposition 3.9 (Complete, Non-redundant Refinement Operators)

Let \mathcal{L} be a description language which contains \mathcal{AL} . Then there exists a complete and non-redundant \mathcal{L} refinement operator.

PROOF We prove the result by showing that complete operators can be transformed to complete and non-redundant operators. Note that in the following, we will use the role r to create concepts with a certain depth. If N_R does not contain any role, the desired effect can also be achieved by using conjunctions or disjunctions of \top and \perp , but this would render the proof less readable.

We will use the fact that the set of concepts in \mathcal{L} is countably infinite. The countability already follows from the fact that there is just a finite number of concepts with a given length. Hence, we can divide the set of all concepts in finite subsets, where each subset contains all concepts of the same length. We can then start enumerating concepts starting with the subset of concepts of length 1, then

length 2 etc. Thus, there is a bijective function $f : \mathcal{L} \mapsto \mathbf{N}$, which assigns a different number to each concept in \mathcal{L} . We denote the inverse function mapping numbers to concepts by f^{inv} .

We modify a given complete operator ρ defined as $\rho(C) = S_C$, where S_C is defined as a maximal subset of $\{D \mid D \sqsubset C\}$ with $D_1, D_2 \in S_C \implies D_1 \not\sqsubset D_2$. ρ is complete, since given a concept C all more special concepts up to weak equivalence can be reached in one refinement step.

We change ρ in the following way: For any concept C , $\rho(C)$ is modified by changing any element $D \in \rho(C)$ to

$$D \sqcap \forall r. (\underbrace{\top \sqcap \dots \sqcap \top}_{f(C) \text{ times}})$$

We claim that the resulting operator, which we denote by ρ' is complete and non-redundant.

The completeness of ρ' follows from the completeness of ρ , since the construct we have added does not change the semantics (it is equivalent to \top).

To prove non-redundancy, we will first define a refinement operator ρ^{inv} , which maps conjunctions, where the last element is of the form $\forall r. (\top \sqcap \dots \sqcap \top)$, to a single concept:

$$\rho^{\text{inv}} = \begin{cases} \{f^{\text{inv}}(n)\} & \text{if } C \text{ is of the form } C \sqcap \forall r. (\underbrace{\top \sqcap \dots \sqcap \top}_{n \text{ times}}) \\ \emptyset & \text{otherwise} \end{cases}$$

We can see that $D \in \rho'(C)$ implies $\rho^{\text{inv}}(D) = \{C\}$, so ρ^{inv} allows to invert a refinement step of ρ' .

By contradiction, we assume ρ' is redundant. Then, there needs to be a concept C and concepts D_1, D_2 with $D_1 \simeq D_2$, such that there is a refinement chain from C to D_1 and a different refinement chain from C to D_2 . Since we know that D_1 and D_2 are refinements of ρ' , they have to be conjunctions, where the last element is of the form $\forall r. (\top \sqcap \dots \sqcap \top)$ and equal in D_1 and D_2 . This means the previous element in both refinement chains is $\rho^{\text{inv}}(D_1) = \rho^{\text{inv}}(D_2)$ and therefore all elements except the last ones (D_1, D_2) in both chains are uniquely determined. Since for any C , $\rho(C)$ does not contain weakly equal concepts, we get that D_1 and D_2 have to be equal. Therefore, both refinement chains are equal, which is a contradiction.

We can establish the same result for upward refinement by using ρ with $\rho(C) = S_C$, where S_C is defined as a maximal subset of $\{D \mid C \sqsubset D\}$ with $D_1, D_2 \in S_C \implies D_1 \not\sqsubset D_2$ as starting point. The construction of ρ' can be done analogously. ■

Proposition 3.10 (Complete, Non-redundant Operators II)

Let \mathcal{L} be a description language which contains \mathcal{AL} . Let ρ be an arbitrary \mathcal{L} downward refinement operator, where for any concept C , $\rho^*(C)$ contains only finitely many different concepts equivalent to \perp . Then ρ is not complete and non-redundant.

The restriction mentioned in Proposition 3.10 is made in order to disallow purely theoretical constructions, as in the proof of Proposition 3.9, which use syntactic concept extensions that do not alter the semantics of a concept, to ensure non-redundancy. We prove the proposition using a milder, but more technical, restriction.

PROOF Let ρ be a complete downward refinement operator, $C_{\text{up}}, C_{\text{down}}$ be concepts with $C_{\text{down}} \sqsubset C_{\text{up}}$, $\{C \mid C \in \rho^*(C_{\text{up}}), C \equiv C_{\text{down}}\}$ be finite, and S be an infinite set of concepts, which are pairwise incomparable, strictly subsumed by C_{up} and strictly subsuming C_{down} . For instance, we could have $C_{\text{up}} = \top$, $C_{\text{down}} = \perp$ and $S = \{\forall r.A, \forall r.\forall r.A, \dots\}$.

Due to the completeness of ρ , there must be a refinement chain from each concept in S to a concept, which is equivalent to C_{down} . Since there are infinitely many concepts in S , but only finitely many different syntactic representations of C_{down} are reached, there have to be concepts C'_{down}, C_1 , and C_2 with $C'_{\text{down}} \equiv C_{\text{down}}$ and $C_1, C_2 \in S$, such that $C'_{\text{down}} \in \rho^*(C_1)$ and $C'_{\text{down}} \in \rho^*(C_2)$.

Because of $C_1 \not\sqsubseteq C_2$ and $C_2 \not\sqsubseteq C_1$, we know $C_1 \notin \rho^*(C_2)$ and $C_2 \notin \rho^*(C_1)$. Hence, there exists a refinement chain from C_{up} to C_{down} through C_1 and a refinement chain from C_{up} to C_{down} , which goes through C_2 and not through C_1 . Thus, ρ is redundant. ■

Again, a dual result and proof for upward refinement operators can be obtained.

As a consequence of the result, completeness and non-redundancy cannot be combined under reasonable assumptions. Usually, it is desirable to have (weakly) complete operators, but in order to have a full analysis of \mathcal{L} refinement operators we will now also investigate incomplete operators.

Proposition 3.11 (Incomplete Refinement Operators)

Let \mathcal{L} be a description language. Then there exists a finite, proper, and non-redundant \mathcal{L} refinement operator.

PROOF The following operator has the desired properties:

$$\rho(C) = \begin{cases} \{\perp\} & \text{if } C \not\equiv \perp \\ \emptyset & \text{otherwise} \end{cases}$$

It is obviously finite, because it maps concepts to sets of cardinality at most 1. It is non-redundant, because it only reaches the bottom concept and there exists no refinement chain of length greater than 2. It is proper, because all concepts, which are not equivalent to the bottom concept strictly subsume the bottom concept.

The corresponding upward operator is:

$$\phi(C) = \begin{cases} \{\top\} & \text{if } C \not\equiv \top \\ \emptyset & \text{otherwise} \end{cases}$$

The arguments for its finiteness, properness, and non-redundancy are analogous to the downward case. ■

We can now summarise the results we have obtained so far.

Theorem 3.12 (Properties of Refinement Operators (I))

Considering the properties completeness, properness, finiteness, and non-redundancy, the following are maximal sets of properties (in the sense that no other of the mentioned properties can be added) of \mathcal{L} refinement operators ($\mathcal{L} \in \{ALL, ALLQ, SHOIN, SROIQ\}$):

1. *{complete, finite}*
2. *{complete, proper}*
3. *{non-redundant, finite, proper}*

All results hold under the mild hypothesis stated in Proposition 3.10.

PROOF The theorem is a consequence of the previous results. We have seen that downward and upward operators allow the same combinations of properties, so it is not necessary to distinguish between them. We make a case distinction:

1. The operator is complete. In this case we cannot add non-redundancy (Proposition 3.10). Finiteness (Proposition 3.3) and properness (Proposition 3.8) can be added, but not both (Proposition 3.7).
2. The operator is not complete. In this case we can add all other properties (Proposition 3.11). ■

3.3 Weak Completeness

A property, which we have not yet considered in detail, is weak completeness. Often weak completeness is sufficient, because it allows to search for a good concept

starting from \top downwards (top-down approach) or from \perp upwards (bottom-up approach).

It will be shown that different results hold when considering weak completeness instead of completeness. As a first observation, we see that the arguments in the proof of Proposition 3.10, which have shown that an \mathcal{L} refinement operator cannot be complete and non-redundant, do no longer apply if we consider weak completeness and non-redundancy.

Proposition 3.13 (Weakly Complete, Non-redundant, Proper Op.)

Let \mathcal{L} be a description language. Then there exists a weakly complete, non-redundant, and proper \mathcal{L} refinement operator.

PROOF The following operator is weakly complete, non-redundant, and proper:

Let S be a maximal subset of $\{C \mid C \not\equiv \top\}$ with $C_1, C_2 \in S \implies C_1 \not\equiv C_2$.

$$\rho(C) = \begin{cases} S & \text{if } C = \top \\ \emptyset & \text{otherwise} \end{cases}$$

Such a set S as used in the definition of the operator indeed exists. It contains one representative of each equivalence class with respect to weak equality of the set $\{C \mid C \not\equiv \top\}$. The operator is proper, since it contains only mappings of the top concept to concepts, which are not equivalent to top. It is non-redundant, because there is no refinement chain of length greater than 1 and all concepts we reach are pairwise not weakly equal. It is weakly complete, because for every concept, which is not equivalent to \top , we can reach an equivalent concept from \top by ρ .

The corresponding upward refinement operator is as follows: Let S be a maximal subset of $\{C \mid C \not\equiv \perp\}$ with $C_1, C_2 \in S \implies C_1 \not\equiv C_2$.

$$\rho(C) = \begin{cases} S & \text{if } C = \perp \\ \emptyset & \text{otherwise} \end{cases}$$

■

The operator just given is obviously not useful in practice, but it suffices for the proof of the proposition.

Proposition 3.14 (Weakly Complete, Non-redundant, Finite Op.)

Let \mathcal{L} be a description language which allows to express conjunction. Then there exists a weakly complete, non-redundant, and finite \mathcal{L} refinement operator.

PROOF The following operator is weakly complete, non-redundant, and finite:

For an arbitrary concept C , let S_C be a maximal subset of $\{D \mid D \sqsubset \top \text{ and } |D| = \text{number of } \top \text{ occurrences in } C\}$ with $C_1, C_2 \in S_C \implies C_1 \not\sqsubset C_2$.

$$\rho(C) = \begin{cases} \underbrace{\{\top \sqcap \dots \sqcap \top\}}_{n+1 \text{ times } \top} \cup S_C & \text{if } C = \underbrace{\top \sqcap \dots \sqcap \top}_{n \text{ times } \top} \\ \emptyset & \text{otherwise} \end{cases}$$

The operator is finite, because S_C is finite for any concept C (the number of concepts with a fixed length is finite). It is weakly complete, because every concept C with $C \sqsubset \top$ can be reached from \top . This is done by accumulating \top symbols until we have $|C|$ such symbols and then generating C . The operator is furthermore non-redundant, because obviously for any concept there is exactly one path for reaching the concept via iterated applications of ρ to \top .

The corresponding upward operator is constructed analogously. It works by accumulating \perp symbols instead of \top symbols, and generates concepts which are strictly more general than \perp . ■

Corollary 3.15 (Weakly Complete, Proper, and Finite Operators)

Let \mathcal{L} be any of the description languages \mathcal{ALC} , \mathcal{ALCQ} , \mathcal{SHOIN} , or \mathcal{SROIQ} . Then there exists no weakly complete, finite, and proper \mathcal{L} refinement operator.

PROOF To show this we can use the proof of Proposition 3.7. There we have shown that in a finite and proper \mathcal{L} refinement operator there exists a concept, which cannot be reached from the \top concept. This means that such an operator cannot be weakly complete. ■

The result of the previous observations is that, when requiring only weak completeness instead of completeness, non-redundant operators are possible. The following theorem is the result of the full analysis of the desired properties of \mathcal{L} refinement operators and the main result of this chapter.

Theorem 3.16 (Properties of Refinement Operators (II))

Considering the properties completeness, weak completeness, properness, finiteness, and non-redundancy the following are maximal sets of properties (in the sense that no other of the mentioned properties can be added) of \mathcal{L} refinement operators ($\mathcal{L} \in \{\mathcal{ALC}, \mathcal{ALCQ}, \mathcal{SHOIN}, \mathcal{SROIQ}\}$):

1. *{weakly complete, complete, finite}*
2. *{weakly complete, complete, proper}*

3. $\{\text{weakly complete, non-redundant, finite}\}$
4. $\{\text{weakly complete, non-redundant, proper}\}$
5. $\{\text{non-redundant, finite, proper}\}$

All results hold under the mild hypothesis stated in Proposition 3.10.

PROOF We can do a similar case distinction as in Theorem 3.12. The first case (complete operator) is analogous except that obviously a complete operator is also weakly complete. For the second case (operator is not complete) we can make a simple case distinction again:

1. The operator is weakly complete. Propositions 3.13 and 3.14 have shown that weakly complete operators can be non-redundant and proper as well as non-redundant and finite. Proposition 3.15 shows that finiteness and properness cannot be combined, so these sets of properties are maximal.
2. The operator is not weakly complete. In this case we can add all remaining properties (Proposition 3.11), i.e. non-redundancy, finiteness, and properness. ■

Note that the restriction of Theorems 3.12 and 3.16 to the languages \mathcal{ALC} , \mathcal{ALCQ} , \mathcal{SHOIN} , and \mathcal{SROIQ} is caused by Proposition 3.7. The result carries over to other DLs, but might require adapting the proofs. We have provided the formal proofs for these four languages, because they are considered to be fundamental, including the DLs underlying OWL and OWL 2.

Remark 3.17 (Subsumption with Respect to a TBox)

Instead of using subsumption (\sqsubseteq) as an ordering over concepts, we can also use subsumption with respect to a TBox \mathcal{T} ($\sqsubseteq_{\mathcal{T}}$). Theorem 3.16 will also hold in this case. In all negative results, i.e. Propositions 3.7, 3.10, Corollary 3.15, we can consider subsumption with respect to an empty TBox as example. In all positive results, i.e. Propositions 3.3, 3.8, 3.11, 3.13, 3.14, we can rewrite the example operators by replacing \sqsubseteq with $\sqsubseteq_{\mathcal{T}}$. □

Remark 3.18 (Weak Equality)

In the definition of redundancy, we used weak equality. Theorem 3.16 also holds if we consider syntactic equality. Proposition 3.9 can be simplified. Proposition 3.10 does not need to be changed. In Propositions 3.13 and 3.14 it is straightforward to modify the refinement operators used as positive examples. □

Remark 3.19 (\mathcal{EL} Family of Description Logics)

The \mathcal{EL} description logic is investigated in Section 4.2. The language \mathcal{EL} is promising due to its tractable reasoning algorithms. Indeed, we will show that \mathcal{EL} allows for ideal refinement. □

4 Designing Refinement Operators

After the thorough theoretical investigations in Chapter 3, we continue with the design of actual refinement operators. We present two different operators: The first operator is very expressive and aims at supporting most OWL structures. As a consequence, it cannot be ideal as we have proven previously. The second operator supports the lightweight description logic \mathcal{EL} . We will show that it is indeed possible to define an ideal operator for this language. Both operators incorporate background knowledge to traverse the search space efficiently.

4.1 A Complete OWL Refinement Operator

We will define the refinement operator ρ using Theorem 3.16 as starting point. The theorem provides five possible maximal property combinations a refinement operator for description logics can have. We need to decide which of these five combinations appears to be the one which is most promising in practice. For this, we have to bear in mind that the strongest theoretically possible property combination is not necessarily the most suitable one for implementation, as the absence of some theoretical properties is more severe than the absence of others. Indeed, it appears reasonable that it is better not to enforce some properties which would be computationally expensive, if at the same time we can algorithmically limit the negative impact this absence may have. We look at each of the properties in turn.

Concerning (weak) completeness, we consider this a very important property, since an incomplete operator may fail to converge at all and thus may not return a solution even if one exists. The fifth property combination from Theorem 3.16 thus appears to be unfavorable.

Concerning finiteness, we will see later in Section 5.1, that having an infinite operator is less critical from a practical perspective, in the sense that this issue can be handled well algorithmically. So it is preferable not to impose finiteness, which allows us to develop a proper operator. This leaves us with the second and fourth property combinations from Theorem 3.16 to choose from.

As for non-redundancy, this appears to be very difficult to achieve for more complex operators than the one in Proposition 3.13, which is not useful in practice as it does not structure the search space at all. Consider, for example, the concept $A_1 \sqcap A_2$ which can be reached from \top via the chain $\top \rightsquigarrow A_1 \rightsquigarrow A_1 \sqcap A_2$. For non-redundancy, the operator would need to make sure that this concept cannot be reached via the chain $\top \rightsquigarrow A_2 \rightsquigarrow A_2 \sqcap A_1$. While there are methods to

handle this in such simple cases via normal forms, it becomes more complex for arbitrarily deeply nested structures, where even applying the same replacement leads to redundancy. In the following example, A_1 is replaced by $A_1 \sqcap A_2$ twice in different order in each chain:

$$\top \rightsquigarrow \forall r_1.A_1 \sqcup \forall r_2.A_1 \rightsquigarrow \forall r_1.A_1 \sqcup \forall r_2.(A_1 \sqcap A_2) \rightsquigarrow \forall r_1.(A_1 \sqcap A_2) \sqcup \forall r_2.(A_1 \sqcap A_2)$$

$$\top \rightsquigarrow \forall r_1.A_1 \sqcup \forall r_2.A_1 \rightsquigarrow \forall r_1.(A_1 \sqcap A_2) \sqcup \forall r_2.A_1 \rightsquigarrow \forall r_1.(A_1 \sqcap A_2) \sqcup \forall r_2.(A_1 \sqcap A_2)$$

To avoid this, an operator would need to regulate when A_1 can be replaced by $A_1 \sqcap A_2$, which appears not to be achievable by syntactic replacement rules. At the same time, we will see in Section 5.1.1 that we can use a computationally inexpensive redundancy check which our experiments have shown to be sufficiently useful in practice.

The arguments just given leave us with the second property combination from Theorem 3.16. And indeed, reasonable weakly complete operators are often also automatically complete. Consider, for example, the situation where a weakly complete operator ρ allows to refine a concept C to $C \sqcap D$ with some $D \in \rho(\top)$. Then it turns out that this operator is already complete (see proof of Proposition 4.7). This observation points us again to the second property combination from Theorem 3.16, which we have already found out to be the most feasible one for developing a refinement operator in our setting.

Summarising, we choose to develop a weakly complete, complete, and proper refinement operator, and will show that we can handle issues arising from infinity and redundancy well algorithmically. This will be detailed in the description of the OCEL learning algorithm in Section 5.1.

We proceed as follows: First, we define a refinement operator and prove its completeness. We then extend it to a complete and proper operator.

4.1.1 Definition of the Operator

We now define the operator and prove that it is indeed a downward refinement operator. For each $A \in N_C$, we define (*sh* stands for subsumption hierarchy):

$$sh_{\downarrow}(A) = \{A' \in N_C \mid A' \sqsubset A, \text{ there is no } A'' \in N_C \text{ with } A' \sqsubset_{\mathcal{T}} A'' \sqsubset_{\mathcal{T}} A\}$$

$sh_{\downarrow}(\top)$ is defined analogously for \top instead of A . $sh_{\uparrow}(A)$ is defined analogously for going upward in the subsumption hierarchy.

We do the same for roles, i.e. :

$$sh_{\downarrow}(r) = \{r' \mid r' \in N_R, r' \sqsubset r, \text{ there is no } r'' \in N_R \text{ with } r' \sqsubset_{\mathcal{T}} r'' \sqsubset_{\mathcal{T}} r\}$$

$domain(r)$ denotes the domain of a role r and $range(r)$ the range of a role r . To recapitulate from Section 2.1.4, a range axiom links a role to a concept. It asserts that the role fillers must be instances of a given concept. Analogously, domain axioms restrict the first argument of role assertions to a concept.

We define:

$$ad(r) = \begin{array}{l} \text{an } A \text{ with } A \in \{\top\} \cup N_C \text{ and } domain(r) \sqsubseteq A \\ \text{and there does not exist an } A' \text{ with } domain(r) \sqsubseteq A' \sqsubset A \end{array}$$

$ar(r)$ is defined analogously using *range* instead of *domain*. ad stands for atomic domain and ar stands for atomic range. We assign exactly one atomic concept as domain/range of a role. Since using atomic concepts as domain and range is very common, $domain$ and ad as well as $range$ and ar will usually coincide.

The set app_B of applicable properties with respect to an atomic concept B is defined as:

$$app_B = \{r \mid r \in N_R, ad(r) = A, A \sqcap B \neq \perp\}$$

To give an example, for the concept **Person**, we have that the role **hasChild** with $ad(\text{hasChild}) = \text{Person}$ is applicable, but the role **hasAtom** with $ad(\text{hasAtom}) = \text{ChemicalCompound}$ is not applicable (assuming **Person** and **ChemicalCompound** are disjoint). We will use this to restrict the search space by ruling out unsatisfiable concepts. The index B describes the context in which the operator is applied, e.g. $\top \rightsquigarrow \text{Person}$ is a refinement step of ρ . However, $\exists \text{hasAtom}.\top \rightsquigarrow \exists \text{hasAtom}.\text{Person}$ is not a refinement step of ρ assuming $ar(\text{hasAtom})$ and **Person** are disjoint.

The set of most general applicable roles mgr_B with respect to a concept B is defined as:

$$mgr_B = \{r \mid r \in app_B, \text{ there is no } r' \text{ with } r \sqsubset r', r' \in app_B\}$$

M_B with $B \in \{\top\} \cup N_C$ is defined as the union of the following sets:

- $\{A \mid A \in N_C, A \sqcap B \neq \perp, A \sqcap B \neq B, \text{ there is no } A' \in N_C \text{ with } A \sqsubset A'\}$
- $\{\neg A \mid A \in N_C, \neg A \sqcap B \neq \perp, \neg A \sqcap B \neq B, \text{ there is no } A' \in N_C \text{ with } A' \sqsubset A\}$
- $\{\exists r.\top \mid r \in mgr_B\}$
- $\{\forall r.\top \mid r \in mgr_B\}$

The operator ρ is defined in Figure 4.1. Note that ρ delegates to an operator ρ_B with $B = \top$ initially. B is set to the atomic range of roles contained in the input concept when the operator recursively traverses the structure of the concept. The index B in the operator (and the set M above) is used to rule out concepts which are disjoint with B .

We use the following notions for different kinds of refinement steps in ρ :

1. \sqcap : add an element conjunctively (cases 3, 4, 5, 6, 8 in the definition of ρ_B in Figure 4.1)
2. \top : refine the top concept (case 2 in the definition of ρ_B in Figure 4.1)

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

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

 Figure 4.1: Definition of the refinement operator ρ .

3. $\overset{A}{\rightsquigarrow}$: refine an atomic concept (case 3 in the definition of ρ_B in Figure 4.1)
4. $\overset{\neg A}{\rightsquigarrow}$: refine a negated atomic concept (case 4 in the definition of ρ_B in Figure 4.1)
5. $\overset{r}{\rightsquigarrow}$: refine a role (cases 5, 6 in the definition of ρ_B in Figure 4.1)

If a concept is refined to some other concept using ρ , exactly one of these five steps is performed. We assume that conjunctions are never nested in conjunctions and disjunctions are never nested in disjunctions, e.g. $A_1 \sqcap (A_2 \sqcap \exists r.\top)$ is written as $A_1 \sqcap A_2 \sqcap \exists r.\top$ instead.

Example 4.1 (ρ refinements)

Since the operator is not easy to understand at first glance, we provide some examples. Let the following knowledge base be given:

$$\begin{aligned} \mathcal{K} = \{ & \text{Man} \sqsubseteq \text{Person}; \text{Woman} \sqsubseteq \text{Person}; \\ & \text{SUV} \sqsubseteq \text{Car}; \text{Limo} \sqsubseteq \text{Car}; \text{Person} \sqcap \text{Car} \equiv \perp; \\ & \text{domain}(\text{hasOwner}) = \text{Car}; \text{range}(\text{hasOwner}) = \text{Person} \} \end{aligned}$$

Then the following refinements of \top exist:

$$\begin{aligned} \rho(\top) = \{ & \text{Car}, \text{Person}, \neg \text{Limo}, \neg \text{SUV}, \neg \text{Woman}, \neg \text{Man}, \\ & \exists \text{hasOwner}.\top, \forall \text{hasOwner}.\top, \text{Car} \sqcup \text{Car}, \text{Car} \sqcup \text{Person}, \dots \} \end{aligned}$$

This illustrates how the set M_\top is constructed. Note that refinements like $\text{Car} \sqcup \text{Car}$ are incorporated in order to reach e.g. $\text{SUV} \sqcup \text{Limo}$ later in a possible refinement chain. The concept $\text{Car} \sqcap \exists \text{hasOwner}.\text{Person}$ has the following refinements:

$$\begin{aligned} \rho(\text{Car} \sqcap \exists \text{hasOwner}.\text{Person}) = \{ & \text{Car} \sqcap \exists \text{hasOwner}.\text{Man}, \\ & \text{Car} \sqcap \exists \text{hasOwner}.\text{Woman}, \\ & \text{SUV} \sqcap \exists \text{hasOwner}.\text{Person}, \\ & \text{Limo} \sqcap \exists \text{hasOwner}.\text{Person}, \dots \} \end{aligned}$$

Note the traversal of the subsumption hierarchy, e.g. Car is replaced by SUV . \square

Proposition 4.2 (Downward Refinement of ρ)

ρ is an \mathcal{ALC} downward refinement operator.

PROOF We have to show that $D \in \rho(C)$ implies $D \sqsubseteq_{\mathcal{T}} C$. We show $D \sqsubseteq C$, i.e. consider an empty TBox. Since $D \sqsubseteq C$ implies $D \sqsubseteq_{\mathcal{T}} C$ by the definition of subsumption, the desired result follows.

We prove by structural induction of \mathcal{ALC} concepts in negation normal form. Obviously, in all cases where $D = C \sqcap C'$, i.e. C is extended conjunctively by a concept C' we have $D \sqsubseteq C$, so these cases are ignored.

- $C = \perp$: $D \in \rho(C)$ is impossible, because $\rho(\perp) = \emptyset$.
- $C = \top$: $D \sqsubseteq C$ is trivially true.
- $C = A$ ($A \in N_C$): $D \in \rho(C)$ implies that D is also an atomic concept and $D \sqsubseteq_{\mathcal{T}} C$. Thus $D \sqsubseteq C$.
- $C = \neg A$: $D \in \rho(C)$ implies that D is of the form $\neg A'$ with $A \sqsubseteq_{\mathcal{T}} A'$. $A \sqsubseteq_{\mathcal{T}} A'$ implies $\neg A' \sqsubseteq_{\mathcal{T}} \neg A$ by the semantics of negation. Thus, $D \sqsubseteq C$.

- $C = \exists r.C'$: $D \in \rho(C)$ implies that D is of the form $\exists r.D'$ or $\exists s.C'$. For the former case, we have $D' \sqsubseteq C'$ by induction. (For existential restrictions $\exists r.E \sqsubseteq \exists r.E'$ if $E \sqsubset E'$ holds in general.) For the latter case, we obviously have $\exists s.C' \sqsubseteq \exists r.C'$, because $s \sqsubset r$.
- $C = \forall r.C'$: This case is analogous to the previous one. Here we use the results $\forall r.E \sqsubseteq \forall r.E'$ ($E \sqsubset_{\mathcal{T}} E'$) and $\forall s.C' \sqsubseteq \forall r.C'$ ($s \sqsubset r$).
- $C = C_1 \sqcap \dots \sqcap C_n$: In this case one element of the conjunction is refined, so $D \sqsubseteq C$ follows by induction.
- $C = C_1 \sqcup \dots \sqcup C_n$: In this case one element of the disjunction is refined, so $D \sqsubseteq C$ follows by induction.

Hence, a refinement of ρ is never more general than the input concept. ■

A distinguishing feature of ρ compared to other DL refinement operators, for instance those in [Badea and Nienhuys-Cheng, 2000, Esposito et al., 2004], is that it makes use of the subsumption and role hierarchy, e.g. for concepts $A_2 \sqsubset A_1$, we reach A_2 via $\top \rightsquigarrow A_1 \rightsquigarrow A_2$. This way, we can stop the search if A_1 is already too weak and, thus, make better use of TBox knowledge. The operator also uses domain and range of roles to reduce the search space. This is similar to mode declarations in Aleph, Progol, and other ILP programs. However, in DL knowledge bases and OWL ontologies, domain and range are usually explicitly given, so there is no need to define them manually. Overall, the operator supports more structures than those in [Badea and Nienhuys-Cheng, 2000, Esposito et al., 2004] and tries to intelligently incorporate background knowledge. Section 4.1.4 describes further extensions of the operator, which increase its expressivity such that it can handle most OWL class expressions.

Note that ρ is infinite. The reason is that the set M_B is infinite and, furthermore, we put no bound on the number of elements in the disjunctions, which are refinements of the top concept. Another point is that the operator requires reasoner requests for calculating M_B . However, the number of requests is fixed, so – assuming the results of those requests are cached – the reasoner is only needed in an initial phase, i.e. during the first calls to the refinement operator. This means that, apart from this initial phase, the refinement operator performs only syntactic rewriting rules.

4.1.2 Completeness of the Operator

To investigate the completeness of the operator, we define a set S_{\downarrow} of \mathcal{ALC} concepts in negation normal form as follows:

Definition 4.3 (S_{\downarrow})

We define $S_{\downarrow} = S'_{\downarrow} \cup \{\perp\}$, where S'_{\downarrow} is defined as follows:

1. If $A \in N_C$ then $A \in S'_\downarrow$ and $\neg A \in S'_\downarrow$.
2. If $r \in N_R$ then $\forall r.\perp \in S'_\downarrow$, $\forall r.\top \in S'_\downarrow$, $\exists r.\top \in S'_\downarrow$.
3. If C, C_1, \dots, C_m are in S'_\downarrow then the following concepts are also in S'_\downarrow :
 - $\exists r.C$ if $C \sqcap ar(r) \not\equiv \perp$
 - $\forall r.C$ if $C \sqcap ar(r) \not\equiv \perp$
 - $C_1 \sqcap \dots \sqcap C_m$
 - $C_1 \sqcup \dots \sqcup C_m$ if for all i ($1 \leq i \leq m$) C_i is not of the form $D_1 \sqcap \dots \sqcap D_n$ where all D_j ($1 \leq j \leq n$) are of the form $E_1 \sqcup \dots \sqcup E_p$. \square

In S'_\downarrow , we do not use the \top and \perp symbols directly and we make a restriction on disjunctions, i.e. we do not allow that elements of a disjunction are conjunctions, which in turn only consist of disjunctions. It can be shown that for any \mathcal{ALC} concept there exists an equivalent concept in S_\downarrow .

Lemma 4.4 (Adequacy of S_\downarrow)

For any \mathcal{ALC} concept C there exists a concept $D \in S_\downarrow$ such that $D \equiv C$.

PROOF We assume C is in negation normal form. The restriction $C \sqcap ar(r) \not\equiv \perp$ for concepts of the form $\exists r.C$ and $\forall r.C$ does not exclude any relevant concepts, because we have $\exists r.C \equiv \exists r.(C \sqcap ar(r))$ and $\forall r.C \equiv \forall r.(C \sqcap ar(r))$ in general. Replacing $C \sqcap ar(r)$ by \perp yields $\exists r.\perp \equiv \perp$ and $\forall r.\perp$. Both, \perp and $\forall r.\perp$, are already in S_\downarrow .

The proof consists of three steps: First, we will eliminate \top symbols unless they occur in existential restrictions (because in Definition 4.3 $\exists r.\top$ is used in the induction base opposed to using \top directly). After that, we do something similar with the bottom symbol. In a third step we will eliminate disjunctions violating the criterion in Definition 4.3. After these three steps, we obtain a concept, which is in S_\downarrow .

We eliminate \top -symbols by applying the following rewrite rules:

$$\begin{aligned}
 C_1 \sqcap \dots \sqcap C_{i-1} \sqcap \top \sqcap C_{i+1} \sqcap \dots \sqcap C_n &\rightarrow C_1 \sqcap \dots \sqcap C_{i-1} \sqcap C_{i+1} \sqcap \dots \sqcap C_n \\
 C_1 \sqcup \dots \sqcup C_{i-1} \sqcup \top \sqcup C_{i+1} \sqcup \dots \sqcup C_n &\rightarrow \top \\
 \forall r.\top &\rightarrow \top
 \end{aligned}$$

Obviously, these \top -elimination steps preserve equivalence. We exhaustively apply these steps (since every step reduces the length of the concept there can be only finitely many such steps) to get a concept C' . Note that $C' \neq \top$ (otherwise $C' \equiv C \equiv \top$) and in C' the top concept only appears in existential restrictions, i.e. in the form $\exists r.\top$.

\perp symbols are eliminated by the following rewrite rules:

$$\begin{aligned} C_1 \sqcap \dots \sqcap C_{i-1} \sqcap \perp \sqcap C_{i+1} \sqcap \dots \sqcap C_n &\rightarrow \perp \\ C_1 \sqcup \dots \sqcup C_{i-1} \sqcup \perp \sqcup C_{i+1} \sqcup \dots \sqcup C_n &\rightarrow C_1 \sqcup \dots \sqcup C_{i-1} \sqcup C_{i+1} \sqcup \dots \sqcup C_n \\ \exists r.\perp &\rightarrow \perp \end{aligned}$$

These steps also preserve equivalence. After exhaustively applying these steps we either get the \perp symbol itself (which is in S_\perp) or the \perp symbol only appears in universal restrictions, i.e. in the form $\forall r.\perp$.

Next we have to eliminate disjunctions, which do not satisfy Definition 4.3. Say we have such a disjunction $C_1 \sqcup \dots \sqcup C_m$. Then there is a C_i ($1 \leq i \leq m$), which is a conjunction consisting only of disjunctions. Without loss of generality we assume $i = 1$ (the order of elements in a disjunction is not important), i.e. we can write $C_1 = D_1 \sqcap \dots \sqcap D_n$ and $D_1 = E_1 \sqcup \dots \sqcup E_p$. This means we can apply the following equivalence preserving rewriting rule:

$$\begin{aligned} ((E_1 \sqcup \dots \sqcup E_p) \sqcap D_2 \sqcap \dots \sqcap D_n) \sqcup C_2 \sqcup \dots \sqcup C_m &\rightarrow \\ (E_1 \sqcap D_2 \sqcap \dots \sqcap D_n) \sqcup \dots \sqcup (E_p \sqcap D_2 \sqcap \dots \sqcap D_n) \sqcup C_2 \sqcup \dots \sqcup C_m \end{aligned}$$

Note that E_i ($1 \leq i \leq p$) cannot be a disjunction. Let C'_1 be the replacement of C_1 after applying the rewriting rule. Obviously, C'_1 is no more a disjunction where an element is a conjunction of disjunctions (because any E_i is not a disjunction). If we apply this rule to all applicable C_i ($1 \leq i \leq m$), then we obtain a concept C'' equivalent to $C_1 \sqcup \dots \sqcup C_m$, which is in S_\perp .

Hence, we have shown that we can construct a concept $C'' \equiv C' \equiv C$ with $C'' \in S_\perp$, which completes the proof. \blacksquare

In the next lemma, we can use the previous result to restrict the attention to S'_\perp .

Lemma 4.5 (Weak Completeness with Domain B)

For any concept B with $B \in \{\top\} \cup N_C$, any \mathcal{ALC} concept $C \in S'_\perp$ with $C \sqsubseteq B$, which satisfies the following restriction, can be reached from \top by ρ_B :

- C is of the form $C_1 \sqcup \dots \sqcup C_m$ and for all i ($1 \leq i \leq m$) $C_i \sqcap B \neq \perp$
- C is not of the form $C_1 \sqcup \dots \sqcup C_m$ and $C \sqcap B \neq \perp$

PROOF We prove by induction over the structure of concepts in S'_\perp . Parts of the rather involved proof are only sketched.

- Induction Base: An atomic concept A can be reached from \top by a refinement chain of the following form:

$$\top \xrightarrow{\top} A_1 \xrightarrow{A} \dots \xrightarrow{A} A_n \xrightarrow{A} A'$$

The operator descends the subsumption hierarchy. Since A' is more special than A , we can always reach it (unless A and A' are disjoint, which is excluded by the induction hypothesis). If B is an atomic concept, then the definition of M_B ensures that A_1 is a concept, which is subsumed by B . Since there are only finitely many atomic concepts, A will be reached in a finite number of steps. Negated atomic concepts can be handled analogously.

$\forall r.\perp$ can be reached by descending the subsumption and role hierarchy:

$$\top \xrightarrow{\top} \forall s.\top \xrightarrow{r} \dots \xrightarrow{r} \forall r.\top \rightsquigarrow \forall r.A_1 \rightsquigarrow \dots \rightsquigarrow \forall r.A_n \rightsquigarrow \forall r.\perp$$

To reach $\exists r.\top$, the following chain can be used:

$$\top \xrightarrow{\top} \exists s.\top \xrightarrow{r} \dots \xrightarrow{r} \exists r.\top$$

$\forall r.\top$ can be handled analogously.

• Induction Step:

- $\exists r.C$: We have $\top \xrightarrow{\top} \exists s.\top \xrightarrow{r} \dots \xrightarrow{r} \exists r.\top$ and by induction we can reach C with $C \sqcap B \not\equiv \perp$ from \top by ρ_A where $A = ar(r)$.
- $\forall r.C$: Analogously to $\exists r.C$.
- $C_1 \sqcap \dots \sqcap C_m$: We know $C_1 \sqcap \dots \sqcap C_m \sqcap A \not\equiv \perp$, which implies $C_i \sqcap A \not\equiv \perp$ for all i ($1 \leq i \leq m$). This means we know that C_1 can be reached from \top using ρ_B by induction. Thus, we can first refine to C_1 and then add all other concepts to the conjunction stepwise:

$$\top \rightsquigarrow^* C_1 \rightsquigarrow^* C_1 \sqcap C_2 \rightsquigarrow^* C_1 \sqcap \dots \sqcap C_m$$

Note that ρ does not allow to extend a conjunction directly, but instead in all other concept structures the operator allows to append an element conjunctively, e.g. $C_1 \sqcap C_2$ can be refined to $C_1 \sqcap (C_2 \sqcap D) \equiv C_1 \sqcap C_2 \sqcap D$ with $D \in \rho_B(\top)$.

- $C_1 \sqcup \dots \sqcup C_m$: We have to show $C_i \in \psi^*(m)$ for an $m \in M_B$. We do this by a case distinction on the structure of C_i for an arbitrary i ($1 \leq i \leq m$).

If C_i is an atomic concept A : In this case, we pick an atomic concept A_1 (see point 1 in the definition of M_B on page 62) and refine it:

$$A_1 \xrightarrow{A} \dots \xrightarrow{A} A_n \xrightarrow{A} A$$

This works exactly as in the case $C = A$ above.

Similarly, if C_i is of the form $\neg A$, we pick a concept $\neg A_2$ according to point 2 in the definition of M_B and refine to $\neg A$. The cases $\exists r.\top$, $\forall r.\top$, $\forall r.\perp$, $\exists r.C$, $\forall r.C$ can be handled analogously.

If C_i is of the form $D_1 \sqcap \dots \sqcap D_n$, then we know that according to the definition of S'_\downarrow there is a D_j which is not a disjunction (and obviously not a conjunction because C_i is a conjunction). This means that D_j can be handled by any of the previous cases. Having refined to D_j , we can then add all other D_k ($1 \leq k \leq n, k \neq j$) to the conjunction.

Summed up, we can refine to C by picking the right concept in M_B for each element of the disjunction and then refining further:

$$\begin{aligned} \top &\rightsquigarrow E_1 \sqcup \dots \sqcup E_m \quad (E_l \in M_B, 1 \leq l \leq m) \\ &\rightsquigarrow^* C_1 \sqcup E_2 \sqcup \dots \sqcup E_m \\ &\rightsquigarrow C_1 \sqcup C_2 \sqcup E_3 \sqcup \dots \sqcup E_m \\ &\rightsquigarrow \dots \rightsquigarrow C_1 \sqcup \dots \sqcup C_m \end{aligned} \quad \blacksquare$$

This allows us to show weak completeness by proving that every element in S_\downarrow can be reached from \top by ρ .

Proposition 4.6 (Weak Completeness of ρ)

ρ is weakly complete.

PROOF We have to show that for any concept C with $C \sqsubseteq_{\mathcal{T}} \top$ a concept E with $E \equiv C$ can be reached from \top by ρ . Due to Lemma 4.4, it is sufficient to show that all concepts in S_\downarrow can be reached from \top by ρ .

Lemma 4.5 proves that we can reach all concepts in S'_\downarrow from \top using ρ_\top , because using $B = \top$ the restriction made in this lemma, is always satisfied:

- If C is not a disjunction: $C \sqcap \top \not\equiv \perp$ is always true unless $C \equiv \perp$, but \perp can be reached from \top using ρ (see below).
- If C is a disjunction: $C_i \sqcap \top \not\equiv \perp$ is always true unless $C_i \equiv \perp$, but in this case C is not in S_\downarrow (\perp cannot be an element of a disjunction in S_\downarrow).

The only element in S_\downarrow , which is not in S'_\downarrow is \perp , which can be reached in one refinement step from \top using ρ . Therefore, all concepts in S_\downarrow can be reached from \top using ρ . ■

Using this, we can prove completeness.

Proposition 4.7 (Completeness of ρ)

ρ is complete.

PROOF Let C and D be arbitrary \mathcal{ALC} concepts in S_1 with $C \sqsubset_{\mathcal{T}} D$. To prove completeness of ρ , we have to show that there exists a concept E with $E \equiv C$ and $E \in \rho^*(D)$. $E = D \sqcap C$ satisfies this property. We obviously have $E = D \sqcap C \equiv C$, because of $C \sqsubset_{\mathcal{T}} D$. We know that ρ allows to extend concepts conjunctively by refinements of the top concept. Hence, we know that $D \sqcap C$ can be reached from D for any concept C by the weak completeness result for ρ . Thus, ρ is complete. ■

4.1.3 Achieving Properness

The operator ρ is not proper, for instance it allows the following refinements:

$$\top \rightsquigarrow \exists r.\top \sqcup \forall r.\top \rightsquigarrow \exists r.\top \sqcup \forall r.A_1 \rightsquigarrow \exists r.A_2 \sqcup \forall r.A_1$$

In this chain, the first three concepts are equivalent. One could try to modify ρ , such that it becomes proper by disallowing certain refinement steps of ρ . However, the last refinement can be reached only via improper refinement steps, hence this strategy does not work. Meaningful modifications of the operator are likely to lead to incompleteness as e.g. $\exists r.A_2 \sqcup \forall r.A_1$ would need to be reached from \top (Proposition 3.8 shows that operators can be complete and proper, but such operators do not structure the search space well). Indeed, there is no tractable structural subsumption algorithm for \mathcal{ALC} [Baader et al., 2007a], which indicates that it is hard to define a proper operator just by syntactic rewriting rules.

So, instead of modifying ρ directly, we allow it to be improper, but consider the closure ρ^{cl} of ρ [Badea and Nienhuys-Cheng, 2000].

Definition 4.8 (ρ^{cl})

ρ^{cl} is defined as follows: $D \in \rho^{cl}(C)$ iff there exists a refinement chain

$$C \rightsquigarrow_{\rho} C_1 \rightsquigarrow_{\rho} \dots \rightsquigarrow_{\rho} C_n = D$$

such that $C \not\equiv D$ and $C_i \equiv C$ for $i \in \{1, \dots, n-1\}$. □

ρ^{cl} is proper by definition. It also inherits the completeness of ρ , since we do not disallow any refinement steps, but only check whether they are improper. We already know that ρ is infinite, so it is clear that we cannot consider all refinements of a concept at a time. Therefore, in practice we will always compute all refinements of a concept up to a given length. A flexible learning algorithm, like the OCEL algorithm presented later, will allow this length limit to be increased if necessary. Using this technique, an infinite operator can be handled. However, we have to make sure that all refinements of ρ^{cl} up to a given length are computable in finite time. To show this, we need the following lemma.

Lemma 4.9 (ρ Does not Reduce Length)

$D \in \rho(C)$ implies $|D| \geq |C|$. Furthermore, there are no infinite refinement chains of the form $C_1 \rightsquigarrow_\rho C_2 \rightsquigarrow_\rho \dots$ with $|C_1| = |C_2| = \dots$, i.e. after a finite number of steps we reach a strictly longer concept.

PROOF To show the first statement we need to observe the steps, which are performed by ρ . As mentioned before, ρ can do one of five things in each refinement step:

1. add an element conjunctively ($\overset{\sqcap}{\rightsquigarrow}$)
2. refine the top concept ($\overset{\top}{\rightsquigarrow}$) not including refinements of the form $\top \rightsquigarrow A$
3. refine an atomic concept ($\overset{A}{\rightsquigarrow}$) including refinements of the form $\top \rightsquigarrow A$
4. refine a negated atomic concept ($\overset{\neg A}{\rightsquigarrow}$)
5. refine a role ($\overset{r}{\rightsquigarrow}$)

Steps 1 and 2 result in a concept with greater length (for this reason we excluded $\top \rightsquigarrow A$ from step 2). Step 3 to 5 result in a concept with the same length. This proves the claim made in the first sentence.

The second claim follows from the fact that there is just a finite number of atomic concepts and roles (N_C, N_R are finite) and there are only finitely many occurrences of an atomic concept within any concept. Hence, there are no infinite refinement chains using only steps 3 to 5. Thus, after a finite number of refinements, step 1 or 2 is used, which produces a longer concept. ■

Proposition 4.10 (Computability up to Length n)

For any concept C in negation normal form and any natural number n , the set $\{D \mid D \in \rho^{cl}(C), |D| \leq n\}$ can be computed in finite time.

PROOF Due to Lemma 4.9, we know that for any concept D in the set, there exists an m such that $|D| > |C|$ with $D \in \rho^m(C)$. Obviously, a concept has only finitely many refinements up to a fixed length. If we consider all refinement chains of a concept C by ρ up to length n as a tree, then this tree is finite (there are only finitely many concepts of length $\leq n$ and any such concept can be reached by a finite refinement chain). The set $\{D \mid D \in \rho^{cl}(C), |D| \leq n\}$ is a subset of the nodes of this tree. Hence, it can be computed in finite time. ■

syntax	construct
r	abstract role
b	boolean concrete role
d	double concrete role
$\leq n \ r.C$	max. cardinality restriction
$\geq n \ r.C$	min. cardinality restriction
$b = \text{true}$	exists boolean true value restriction
$b = \text{false}$	exists boolean false value restriction
$d \leq v$	exists double max. restriction ($v \in \mathcal{R}$)
$d \geq v$	exists double min. restriction ($v \in \mathcal{R}$)
syntax	semantics
r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
b	$b^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \{\text{false}, \text{true}\}$
d	$d^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \mathcal{R}$
$\leq n \ r.C$	$(\leq n \ r.C)^{\mathcal{I}} = \{a \mid \text{card}\{b \mid (a, b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\} \leq n\}$
$\geq n \ r.C$	$(\geq n \ r.C)^{\mathcal{I}} = \{a \mid \text{card}\{b \mid (a, b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\} \geq n\}$
$b = \text{true}$	$(b = \text{true})^{\mathcal{I}} = \{a \mid (a, \text{true}) \in b^{\mathcal{I}}\}$
$b = \text{false}$	$(b = \text{false})^{\mathcal{I}} = \{a \mid (a, \text{true}) \in b^{\mathcal{I}}\}$
$d \leq v$	$(d \leq v)^{\mathcal{I}} = \{a \mid \exists v'. (a, v') \in d^{\mathcal{I}} \text{ and } v' \leq v\}$
$d \geq v$	$(d \geq v)^{\mathcal{I}} = \{a \mid \exists v'. (a, v') \in d^{\mathcal{I}} \text{ and } v' \geq v\}$

Table 4.1: Overview of syntax and semantics of ρ extensions. See Table 2.1 for *SHOIN* syntax and semantics. $\text{card}\{\dots\}$ denotes set cardinality.

Due to Proposition 4.10 we can use ρ^{cl} in a learning algorithm. For computing ρ^{cl} up to length n , it is sufficient to apply the operator until a non-equivalent concept is reached. By a straightforward analysis of the refinement steps, one can show that in the worst case after $O(|N_C| \cdot |N_R| \cdot |C|)$ steps a refinement of greater length will be reached, which bounds the complexity of computing the closure.

4.1.4 Cardinality Restrictions and Concrete Role Support

This section describes an extension of the presented refinement operator with cardinality restrictions and support for boolean and double concrete roles (called data properties in OWL). Syntax and semantics of those constructs can be found in Table 4.1. They can be used to construct concepts such as $\text{Person} \sqcap \text{height} \geq 1.85$ (persons taller than 1.85), $\text{Student} \sqcap \geq 3 \text{ hasCar} . \top$ (students with more than 3 cars), $\text{Patient} \sqcap \text{pregnancyTest} = \text{true}$ (patients with positive pregnancy test).

They are described as extensions here, because we wanted to keep the presentation of the refinement operator brief. Furthermore, the operator becomes incomplete if those extensions are included. For instance, when enabling double concrete role support, **Person** can be refined to $\mathbf{Person} \sqcap \mathbf{height} \geq x$, where x is one of finitely many values determined by analysing the knowledge base (described below). Since the set of real numbers is infinite, this means we cannot – and of course do not want to – reach all concepts of the form $\mathbf{Person} \sqcap \mathbf{height} \geq x$. Thus, the operator is not complete.

To support the constructs listed in Table 4.1, the refinement operator was extended as follows:

Analogously to the already introduced sets N_C , N_R , and N_I in Section 2.1.3, we introduce the following notions: The set N_{CR} stands for the set of all concrete roles, the set N_{BCR} stands for the set of all boolean concrete roles, and the set N_{DCR} stands for the set of all double concrete roles.

The sets mgb for boolean concrete roles and mgd for double concrete roles are defined analogously to mgr (see Section 4.1.1).

Let $values_d$ ($d \in N_{DCR}$) be a list containing the following double numbers in ascending order: $\{t \mid \mathcal{K} \models d(a, t)\}$. $values_d[i]$ denotes the i -th element in this list. $\#splits_d \in \mathcal{N}$ is a user defined parameter of the operator for specifying how many refinement steps should be used to traverse the values of double concrete roles. The list $splits_d$ contains the following double numbers in ascending order:

$$\begin{aligned} \{t_j \mid i = \frac{\#values_d}{\#splits_d + 1}, \\ t = \frac{1}{2}(values_d[\lfloor i \cdot j \rfloor] + values_d[\lfloor i \cdot j \rfloor + 1]) \text{ for } 1 \leq j \leq \#splits_d\} \end{aligned}$$

Again, we use $splits_d[i]$ to refer to the i -th element in this list. Clearly, one can employ different strategies for finding sensible splitting values.

$mf_r = \max_{a \in N_I} |\{b \mid \mathcal{K} \models r(a, b)\}|$ is the maximum number of role fillers of a role r . We use this as upper limit for cardinality restrictions.

The set M_B is extended by the following sets:

- $\{\leq mf_r r. \top \mid r \in mgr_B\}$
- $\{b = true \mid b \in mgb_B\}$
- $\{b = false \mid b \in mgb_B\}$
- $\{d \geq v \mid d \in mgd_B, v = splits_d[\#splits_d]\}$
- $\{d \leq v \mid d \in mgd_B, v = splits_d[1]\}$

Finally, the refinement operator is extended as follows for handling the novel constructs:

$$\rho_B(C) = \begin{cases} \geq 2 \ r.D & \text{if } C = \exists r.D \\ \{\geq n + 1 \ r.D \mid n < \text{mf}_r\} \cup \{\geq n \ r.E \mid E \in \rho_B(D)\} & \text{if } C = \geq n \ r.D \\ \{\leq n - 1 \ r.D \mid n > 1\} \cup \{\leq n \ r.E \mid E \in \rho_B(D)\} & \text{if } C = \leq n \ r.D \\ \emptyset & \text{if } C = (b = \text{true}) \\ \emptyset & \text{if } C = (b = \text{false}) \\ \{d \geq w \mid v = \text{splits}_d[i], i > 1, \\ \quad w = \text{splits}_d[i - 1]\} & \text{if } C = (d \geq v) \\ \{d \leq w \mid v = \text{splits}_d[i], i < \#\text{splits}_d, \\ \quad w = \text{splits}_d[i + 1]\} & \text{if } C = (d \leq v) \end{cases}$$

Intuitively, the modifications have the following effects:

boolean concrete roles: Including boolean concrete roles is straightforward. Occurrences of \top can be refined to e.g. `pregnancyTest = true` and those concepts cannot be further refined.

double concrete roles: Incorporating double concrete roles is slightly more involved. An example refinement chain is $\top \rightsquigarrow \text{height} \geq 1.97 \rightsquigarrow \text{height} \geq 1.92$. The operator starts with a very high value of a concrete role (`height` in this case) when the operator \geq is introduced and then allows this value to be reduced in further refinement steps. The approach employed in the operator is to collect all values of the role in an ordered list and then defining appropriate split values, where the number of splits is configurable. The underlying idea is to avoid too small specialisation steps of the refinement operator. For instance, in a knowledge base containing 10000 different heights of buildings, it is usually not desirable to traverse the height values one by one. Also note that only the order of values is important. This allows learning algorithms to work reasonably well even if the values differ by order of magnitudes, e.g. in case heights of persons and buildings are included in the background knowledge base. In general, the integration of numerical information in inductive systems is challenging (see e.g. [Esposito et al., 2001] which presents a more general framework for the inclusion of numerical data and an approach for determining cut-off points in the context of ILP).

cardinality restrictions: An example of a refinement chain involving cardinality restrictions is $\top \rightsquigarrow \exists r.\top \rightsquigarrow \geq 2 \ r.\top \rightsquigarrow \geq 3 \ r.\top$. Minimum cardinality restrictions are introduced as refinements of existential restrictions. The operator allows to refine the cardinality up to the maximum number of role fillers of an object, which was determined beforehand. To improve efficiency, a manual cardinality limit can be set.

The described extensions were used in the carcinogenesis benchmark (Section 7.2.2). They extend the expressiveness of the target language close to those of OWL class expressions.

4.1.5 Optimisations

In this section improvements of ρ and ρ^{cl} will be presented.

Using $\exists r.(C \sqcup D) \equiv \exists r.C \sqcup \exists r.D$ and $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$:

The equivalences $\exists r.(C \sqcup D) \equiv \exists r.C \sqcup \exists r.D$ and $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ can be used to modify ρ without losing weak completeness.

Disjunctions in ρ are only introduced in refinements of the top concept. The only existential value restrictions in these disjunctions are of the form $\exists r.\top$ for $r \in N_R$. The equivalence $\exists r.(C \sqcup D) \equiv \exists r.C \sqcup \exists r.D$ says that it is not necessary to allow several disjuncts of the form $\exists r.\top$ for a fixed role r , because we can always reach an equivalent concept by only introducing it once. Therefore, we can restrict ρ to produce $\exists r.\top$ only at most once per role as element of the disjunction in the refinement of the top concept, without losing completeness.

ρ allows to refine a concept C by extending it conjunctively. If C is of the form $\forall r.D$ or of the form $C_1 \sqcap \dots \sqcap \forall r.D \sqcap \dots \sqcap C_n$, then we can restrict ρ to disallow adding an element of the form $\forall r.E$ (E is an arbitrary \mathcal{ALC} concept). Again, the resulting operator is still complete.

Similar checks can be done in the refinement steps which refine roles. By using the equalities $\exists r.(C \sqcup D) \equiv \exists r.C \sqcup \exists r.D$ and $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as described, we have reduced the number of possible refinements, but preserved completeness.

Configurable Target Language:

One of the factors determining whether a learning algorithm will be successful for a given task is whether its target language is suitable. ρ can be configured to some extent in this respect by adapting the used refinement operator. In particular, the DL-Learner implementation allows to ignore a specified set of concepts and roles (or conversely use only a specified set of concepts and roles) when trying to solve a problem. It can also be configured to use or ignore the \exists and \forall concept constructors, negation, cardinality restrictions, double concrete roles, boolean concrete roles. These options can be used to incorporate additional knowledge about a problem in the learning algorithm to narrow the search space. Note, that all experiments reported in Section 7 were run without such restrictions.

4.2 An Ideal EL Refinement Operator

We developed a second operator to cover the successful \mathcal{EL} family of description languages and close a gap in the theoretical analysis in Chapter 3. \mathcal{EL} is a light-

Concept constructor	Syntax	Semantics
Top	\top	$\Delta^{\mathcal{I}}$
Concept name	A	$A^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \text{there is } y \in C^{\mathcal{I}} \text{ with } (x, y) \in r^{\mathcal{I}}\}$

Table 4.2: \mathcal{EL} syntax and semantics.

Name	Syntax	Restriction on \mathcal{I}
Concept inclusion	$A \sqsubseteq B$	$A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$
Role inclusion	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
Disjointness	$A \sqcap B \equiv \perp$	$A^{\mathcal{I}} \cap B^{\mathcal{I}} = \emptyset$
Domain	$\text{domain}(r) = A$	$x \in A^{\mathcal{I}}$ for all $(x, y) \in r^{\mathcal{I}}$
Range	$\text{range}(r) = A$	$y \in A^{\mathcal{I}}$ for all $(x, y) \in r^{\mathcal{I}}$

Table 4.3: \mathcal{EL} Knowledge base axioms.

weight DL, but despite its limited expressive power it has proven to be of practical use in many real-world large-scale applications. For example, the Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT) [Bodenreider et al., 2007] and the GENE ONTOLOGY [The Gene Ontology Consortium, 2000] are based on \mathcal{EL} . Since standard reasoning in \mathcal{EL} is polynomial, it is suitable for large ontologies. It should furthermore be mentioned that \mathcal{EL}^{++} , an extension of \mathcal{EL} , is one of three profiles in the new standard ontology language OWL 2.

The operator defined here is quite different, compared to the previously introduced operator ρ as well as other DL operators, in that concepts are viewed as tree structures. Its main advantage is its ideality, which allows to define simple and efficient learning algorithms.

Table 4.2 summarizes syntax and semantics of \mathcal{EL} . For the purpose of defining an \mathcal{EL} refinement operator, we do consider inferred knowledge bases as defined in Table 4.3. This is analogous to the OWL refinement operator, where we also assumed that those structures (subsumption hierarchy, disjoint concepts, domain/range of roles) are inferred from the background knowledge. We will later show that the operator is ideal with respect to knowledge bases containing the mentioned axioms. Please note that for showing ideality of a refinement operator, it would be sufficient to consider an empty knowledge base, but we explicitly use more background knowledge.

Given finite sets of concept names $\mathcal{A}, \mathcal{B} \subseteq N_C$, we write $\mathcal{A} \sqsubseteq \mathcal{B}$ iff for every $B \in \mathcal{B}$ there is some $A \in \mathcal{A}$ such that $A \sqsubseteq B$. We sometimes abuse notation and

write $\mathcal{A} \sqsubseteq B$ instead of $\mathcal{A} \sqsubseteq \{B\}$. We call $\mathcal{A} \subseteq N_C$ *reduced* if there does not exist $\mathcal{B} \subseteq N_C$ with $|\mathcal{B}| < |\mathcal{A}|$ and $\mathcal{A} \equiv \mathcal{B}$.

4.2.1 \mathcal{EL} Trees and Simulation Relations

An important observation is that \mathcal{EL} concepts can be viewed as directed labeled trees, see e.g. [Baader et al., 1999]. This allows for deciding subsumption between concepts in terms of the existence of a simulation relation between the nodes of their corresponding trees. Moreover, the graph approach to \mathcal{EL} concepts allows for a canonical representation of \mathcal{EL} concepts as minimal \mathcal{EL} trees. The latter generalises similar approaches found in the literature, namely “reduced \mathcal{EL} concept terms” [Küsters, 2001] and “minimal XPath tree pattern queries” [Ramanan, 2002]. Some proofs are omitted in this section, since they are mostly a straight-forward generalisation of the proofs found in [Küsters, 2001]. We refer to [Lehmann and Haase, 2009a] for details.

An \mathcal{EL} graph is a directed labeled graph $G = (V, E, \ell)$, where V is the finite set of *nodes*, $E \subseteq V \times N_R \times V$ is the set of *edges*, and $\ell : V \rightarrow \mathcal{P}(N_C)$ is the *labeling function*. We define $V(G) := V$, $E(G) := E$, $\ell(G) := \ell$ and $|G| := |V| + |E|$. For an edge $(v, r, w) \in E$, we call w an (r) -*successor* of v , and v an (r) -*predecessor* of w . Given a node $v \in V$, a labelling function ℓ and $L \subseteq N_C$, we define $\ell[v \mapsto L]$ as $\ell[v \mapsto L](v) := L$ and $\ell[v \mapsto L](w) := \ell(w)$ for all $w \neq v$. Given G and $v \in V(G)$, we define $G[v \mapsto L] := (V(G), E(G), \ell(G)[v \mapsto L])$. We say $v_1 \xrightarrow{r_1} \dots \xrightarrow{r_n} v_{n+1}$ is a *path* of length n from v_1 to v_{n+1} in G iff $(v_i, r_i, v_{i+1}) \in E$ for $1 \leq i \leq n$. A graph G contains a cycle iff there is a path $v \xrightarrow{r_1} \dots \xrightarrow{r_n} v$ in G .

An \mathcal{EL} concept is represented by an \mathcal{EL} *concept tree*, which is a connected finite \mathcal{EL} graph t that does not contain any cycle, has a distinguished node called the *root* of t that has no predecessor, and every other node has exactly one predecessor along exactly one edge. The set of \mathcal{EL} concept trees is denoted by T . In the following, we call an \mathcal{EL} concept tree just a tree. Figure 4.2 illustrates two examples of such trees. Given a tree t , we denote by $\text{root}(t)$ its root. The tree t corresponding to a concept C is defined by induction on $n = \text{rdepth}(C)$. For $n = 0$, t consists of a single node that is labelled with all concepts names occurring in C . For $n > 0$, the root of t is labelled with all concept names occurring on the top-level of C . Furthermore, for each existential restriction $\exists r.D$ on the top-level of C , it has an r -labelled edge to the root of a subtree of t' which corresponds to D . As an example, the tree t corresponding to $A_1 \sqcap \exists r.A_2$ is $t = (\{v_1, v_2\}, \{(v_1, r, v_2)\}, \ell)$ where ℓ maps v_1 to $\{A_1\}$ and v_2 to $\{A_2\}$. By t_\top we denote the tree corresponding to \top . Obviously, the transformation from a concept to a tree can be performed in linear time w.r.t. the size of the concept. Similarly, any tree has a corresponding concept¹, and the transformation can be performed in linear time, too.

Let t, t' be trees, $v \in V(t)$ and assume w.l.o.g. that $V(t) \cap V(t') = \emptyset$. Denote

¹Strictly speaking, t has a set of corresponding concepts, which are all equivalent up to commutativity.

by $t[v \leftarrow (r, t')]$ the tree obtained from plugging t' via an r -edge into the node v of t , i.e. the tree $(V(t) \cup V(t'), E(t) \cup E(t') \cup \{(v, r, \text{root}(t'))\}, \ell \cup \ell')$, where $\ell \cup \ell'$ is the obvious join of the labeling functions of t and t' . By $t(v)$ we denote the subtree at v . Let C be a concept and t the tree corresponding to C . We define $\text{depth}(t) := \text{rdepth}(C)$, and for $v \in V(t)$, $\text{level}(v) := \text{depth}(t) - \text{depth}(t(v))$. Moreover, $\text{onlevel}(t, n)$ is the set of nodes $\{v \mid \text{level}(v) = n\}$ that appear on level n in tree t .

Definition 4.11 (\mathcal{EL} Simulation)

Let $t = (V, E, \ell), t' = (V', E', \ell')$ be trees. A *simulation relation* from t' to t is a binary relation $\mathcal{S} \subseteq V \times V'$ such that if $(v, v') \in \mathcal{S}$ then the following *simulation conditions* are fulfilled:

$$(\text{SC1}) \quad \ell(v) \sqsubseteq \ell'(v')$$

$$(\text{SC2}) \quad \text{for every } (v', r, w') \in E' \text{ there is } (v, r, w) \in E_1 \text{ such that } r \sqsubseteq r' \text{ and } (w, w') \in \mathcal{S} \quad \square$$

We write $t \preceq t'$ if there exists a simulation relation \mathcal{S} from t' to t such that $(\text{root}(t), \text{root}(t')) \in \mathcal{S}$. It is easily checked that (T, \preceq) forms a quasi ordered space, and we derive the relations \simeq and $<$ accordingly. A simulation \mathcal{S} from t' to t is *maximal* if for every simulation \mathcal{S}' from t' to t , $\mathcal{S}' \subseteq \mathcal{S}$. It is not hard to check that \mathcal{S} is unique. Using a dynamic programming approach, the maximal simulation can be computed in $\mathcal{O}(|t| \cdot |t'|)$. Figure 4.2 shows an example of a simulation.

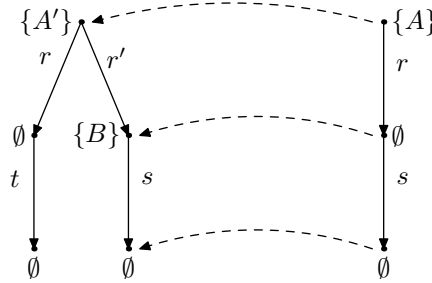


Figure 4.2: A (non-maximal) simulation relation w.r.t. the knowledge base $\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$ from the tree corresponding to $A \sqcap \exists r. \exists s. \top$ to the tree corresponding to $A' \sqcap \exists r. \exists t. \top \sqcap \exists r'. (B \sqcap \exists s. \top)$.

Definition 4.12 (\mathcal{EL} Tree)

Let C be a concept. The tree t corresponding to C is defined by induction on $n = \text{rdepth}(C)$. For $n = 0$, we have that $C = A_1 \sqcap \dots \sqcap A_k$ and define $t := (\{v\}, \emptyset, \ell)$ with $\ell(v) := \{A_1, \dots, A_k\}$. For $n > 0$, $C = A_1 \sqcap \dots \sqcap A_k \sqcap \exists r_1. D_1 \sqcap \dots \sqcap \exists r_m. D_m$ with $\text{rdepth}(D_i) < n, 1 \leq i \leq m$. By the induction hypothesis, for each D_i there exists a tree $t_i = (V_i, E_i, \ell_i), 1 \leq i \leq m$. Without loss of generality assume $V_i \cap V_j = \emptyset, 1 \leq i \neq j \leq m$. Define $t := (V, E, \ell)$ where

- $V := \{v\} \cup \bigcup_{1 \leq i \leq m} V_i$
- $E := \{(v, r_i, \text{root}(t_i)) \mid 1 \leq i \leq m\} \cup \bigcup_{1 \leq i \leq m} E_i$
-

$$\ell(w) := \begin{cases} \{A_1, \dots, A_k\} & \text{if } w = v \\ \ell_i(w) & \text{if } w \in V_i \end{cases} \quad \square$$

Definition 4.13 (\mathcal{EL} Graph of an Interpretation)

Let \mathcal{I} be an interpretation. The \mathcal{EL} graph $\mathcal{G}_{\mathcal{I}} = (V_{\mathcal{I}}, E_{\mathcal{I}}, \ell_{\mathcal{I}})$ corresponding to \mathcal{I} is defined as follows:

- $x \in V_{\mathcal{I}}$ iff $x \in \Delta^{\mathcal{I}}$
- $(x, r, y) \in E_{\mathcal{I}}$ iff $(x, y) \in r^{\mathcal{I}}$
- $A \in \ell_{\mathcal{I}}(x)$ iff $x \in A^{\mathcal{I}}$

□

This definition also allows us to view \mathcal{EL} graphs and in particular \mathcal{EL} trees as interpretations.

Lemma 4.14 (Simulations and Interpretations)

Let C be an \mathcal{EL} concept with the corresponding \mathcal{EL} tree $t = (V, E, \ell)$ with root v , and let \mathcal{I} be an interpretation with $x \in \Delta^{\mathcal{I}}$ and the corresponding \mathcal{EL} graph $\mathcal{G} = (V_{\mathcal{I}}, E_{\mathcal{I}}, \ell_{\mathcal{I}})$. Then $x \in C^{\mathcal{I}}$ iff there exists a simulation \mathcal{S} from v to x .

PROOF The proof is by induction on $d = \text{rdepth}(C)$ in both directions.

(\Rightarrow) For the induction base case, let $d = 0$ and $x \in (A_1 \sqcap \dots \sqcap A_k)^{\mathcal{I}}$. Define $\mathcal{S} := \{(x, v)\}$, which obviously is a simulation. Now for the induction step, let $x \in (A_1 \sqcap \dots \sqcap A_k \sqcap \exists r_1.C_1 \sqcap \dots \sqcap \exists r_m.C_m)^{\mathcal{I}}$. There are $(x, v_i) \in r_i^{\mathcal{I}}$ such that $v_i \in C_i^{\mathcal{I}}$, $(x, r_i, v_i) \in E_C$ and by the induction hypothesis there exist simulations \mathcal{S}_i from v_i to x_i for $1 \leq i \leq m$. Hence, $\mathcal{S} := \bigcup_{1 \leq i \leq m} \mathcal{S}_i \cup \{(x, v)\}$ is a simulation from v to x .

(\Leftarrow) For the induction base case, let $d = 0$ and $C = A_1 \sqcap \dots \sqcap A_k$. For every $A \in \ell(v)$ we have $A' \in \ell_{\mathcal{I}}(x)$ with $A' \sqsubseteq A$, so clearly $x \in C^{\mathcal{I}}$. For the induction step, let $C = A_1 \sqcap \dots \sqcap A_k \sqcap \exists r_1.C_1 \sqcap \dots \sqcap \exists r_m.C_m$ and \mathcal{S} be a simulation from v to x . There are $(v, r_i, v_i) \in E$ and by the simulation conditions, there are also $(x, x_i) \in r_i^{\mathcal{I}}$ and $(x_i, v_i) \in \mathcal{S}$ for $1 \leq i \leq m$. Now \mathcal{S} is a simulation from each v_i to x_i , and hence by the induction hypothesis $x_i \in C_i^{\mathcal{I}}$. Consequently, $x \in C^{\mathcal{I}}$. ■

Lemma 4.15 (Uniqueness of Maximal Simulations)

Given trees t_1, t_2 , the maximal simulation from t_2 to t_1 is unique.

PROOF Suppose there are maximal simulations S_1, S_2 from t_2 to t_1 with $S_1 \not\subseteq S_2$ and $S_2 \not\subseteq S_1$. Observe that $S := S_1 \cup S_2$ is a simulation from t_2 to t_1 , $S_1 \subset S$ and $S_2 \subset S$, which contradicts to the maximality of S_1 and S_2 . ■

The following lemma is proven by induction on $rdepth(D)$. It allows us to decide subsumption between concepts C, D in terms of the existence of a simulation between their corresponding trees t, t' , and moreover to interchange concepts and their corresponding trees. For that reason, the \mathcal{EL} refinement operator presented in the next section will work on trees rather than concepts.

Lemma 4.16 (Subsumption and Simulations)

Let C, D be concept with their corresponding trees t, t' . Then $C \sqsubseteq D$ iff $t \preceq t'$.

PROOF In the following let $x_C = root(t)$ and $x_D = root(t')$.

(\Rightarrow) We show the contrapositive. Assume there does not exist a simulation from x_D to x_C . Now the identity on the vertices of t_C is a simulation from x_C to x_C and hence $x_C \in C^{\mathcal{I}}$, where \mathcal{I} is the interpretation corresponding to t_C . Since there does not exist a simulation from x_D to x_C , the previous lemma gives $x_C \notin D^{\mathcal{I}}$.

(\Leftarrow) Let \mathcal{S} be a simulation from x_D to x_C , and let \mathcal{I} be an interpretation with $y \in C^{\mathcal{I}}$. By the previous lemma, there exists a simulation \mathcal{S}' and the composition $\mathcal{S} \circ \mathcal{S}'$ yields a simulation from x_D to y . Hence $y \in D^{\mathcal{I}}$. ■

The proof of Lemma 4.16 gives rise to Algorithm 1. Given \mathcal{EL} trees t_C and t_D , starting from the leaves of t_D , it computes bottom-up the maximal simulation from t_D to t_C in $\mathcal{O}(|t_C| \cdot |t_D|)$.

Lemma 4.17 (Computation of Maximal Simulation)

Given \mathcal{EL} trees t_C and t_D with roots x_C and x_D , Algorithm 1 computes the maximal simulation \mathcal{S} from t_D to t_C .

PROOF We prove the statement by induction on $n = depth(t_D)$. The induction base case follows obviously. For the induction step, let \mathcal{S}_n be the relation obtained in the algorithm after iterating the outermost **for**-loop n times and let \mathcal{S} be the

Algorithm 1: Computing the maximal simulation

Input: \mathcal{EL} trees t_C, t_D

```

1  $\mathcal{S} := \emptyset;$ 
2 for  $i = 0; i \leq \text{depth}(t_D); i := i + 1$  do
3   forall  $w \in V_D$  with  $\text{depth}(w) = i$  do
4     forall  $v \in V_C$  do
5       if (SC1) and (SC2) hold for  $(v, w)$  then
6          $\mathcal{S} := \mathcal{S} \cup \{(v, w)\}$ 
7 return  $\mathcal{S}$ 

```

relation obtained from the algorithm. It follows from the induction hypothesis that \mathcal{S}_n is the maximal simulation from the subtree of every successor node of x_D to t_1 . Hence for the maximal simulation \mathcal{S}' from t_D to t_C , $\mathcal{S}' \setminus (V(t_C) \times \{x_D\}) \subseteq \mathcal{S}_n$. Now assume $(x_C, x_D) \in \mathcal{S}'$, but $(x_C, x_D) \notin \mathcal{S}$. Then for every r_2 -successor y_D of x_D , there exist an r_1 -successor y_C of x_C with $r_1 \sqsubseteq r_2$, and $(y_C, y_D) \in \mathcal{S}'$. However, $(y_C, y_D) \in \mathcal{S}_n \subseteq \mathcal{S}$ and consequently the pair (x_C, x_D) is also added in the last run of the outermost **for**-loop to \mathcal{S} . Hence, \mathcal{S} is maximal. ■

We can now introduce minimal \mathcal{EL} trees which serve as a canonical representation of equivalent \mathcal{EL} concepts.

Definition 4.18 (Minimal Trees)

Let $t = (V, E, \ell)$ be a tree. We call t *label reduced* if for all $v \in V$, $\ell(v)$ is reduced, i.e. no concept name can be removed from the label without resulting in an inequivalent tree. Moreover, t *contains redundant subtrees* if there are $(v, r, w), (v, r', w') \in E$ with $w \neq w'$, $r \sqsubseteq r'$ and $t(w) \preceq t(w')$. We call t *minimal* if t is label reduced and does not contain redundant subtrees. □

It follows that the minimality of a tree t can be checked in $\mathcal{O}(|t|^2)$ by computing the maximal simulation from t to itself and then checking for each $v \in V(t)$ whether v is label reduced and, using \mathcal{S} , whether v is not the root of redundant subtrees. The set of minimal \mathcal{EL} trees is denoted by T_{\min} .

We close this section with two small lemmas that will be helpful in the next section.

Lemma 4.19 (Finite Number of \mathcal{EL} Trees up to some Depth)

Let T_n be the set of minimal \mathcal{EL} trees up to depth $n \geq 0$. Then $|T_n|$ is finite.

PROOF The proof is by induction on $n = 0$. We have $T_0 = 2^{|N_C|}$. For the induction step, assume T_{n+1} is infinite. Hence, there is a tree $t \in T_{n+1}$ whose

root v has more than $|N_R| \cdot |T_n|$ outgoing edges. Consequently, there are distinct $(v, r, w), (v, r, w') \in E$ such that $t(w) \simeq t(w')$, which contradicts to t being minimal. ■

Lemma 4.20 (No Simulation Between Trees of Different Depth)

Let t, t' be \mathcal{EL} trees with $\text{depth}(t) < \text{depth}(t')$. Then $t \not\sqsubseteq t'$.

PROOF Let $\text{root}(t) = v$, $\text{depth}(t) = n$ and $v \xrightarrow{r_1} \dots \xrightarrow{r_n} v_{n+1}$ be a path of length n in t . Since t' is tree, i.e. an acyclic graph of depth $m < n$, there cannot be $w \in V(t')$ and a relation $\mathcal{S} \subseteq V(t') \times V(t)$ from t to t' such that (w, v_m) and (SC2) from Definition 4.11 holds. ■

4.2.2 Formal Description of the \mathcal{EL} Refinement Operator

In this section, we define an ideal refinement operator. In the first part, we are more concerned with a description of the operator on an abstract level, which allows us to prove its properties. The next part addresses optimisations of the operator that improve its performance in practice.

Definition of the Operator

For simplicity, we subsequently assume the knowledge base to only contain concept and role inclusion axioms. We will sketch in the next section how the remaining restriction axioms can be incorporated in the refinement operator.

The refinement operator ψ , to be defined below, is a function that maps a tree $t \in T_{\min}$ to a subset of T_{\min} . It can be divided into the three base operations *label extension*, *label refinement* and *edge refinement*. Building up on that, the complex operation *attach subtree* is defined. Each such operation takes a tree $t \in T_{\min}$ and a node $v \in V(t)$ as input and returns a set of trees that are refined at node v . Figure 4.3 provides an example.

The base operations are as follows: the operation $el(t, v)$ returns the set of those minimal trees that are derived from t by extending the label of v . Likewise, $rl(t, v)$ is the set of minimal trees obtained from t by refining the label of v . Last, $re(t, v)$ is obtained from t by refining one of the outgoing edges at v . Formally,

- $el(t, v)$: $t' \in el(t, v)$ iff $t' \in T_{\min}$ and $t' = t[v \mapsto (\ell(v) \cup \{A\})]$, where $A \in \max\{B \in N_C \mid \ell(v) \not\sqsupseteq B\}$
- $rl(t, v)$: $t' \in rl(t, v)$ iff $t' \in T_{\min}$ and $t' = t[v \mapsto (\ell(v) \cup \{A\}) \setminus \{B\}]$, where $B \in \ell(v)$, $A \in \max\{A' \in N_C \mid A' \sqsubset B\}$ and there is no $B' \in \ell(v)$ with $B \neq B'$ and $A \sqsubset B'$

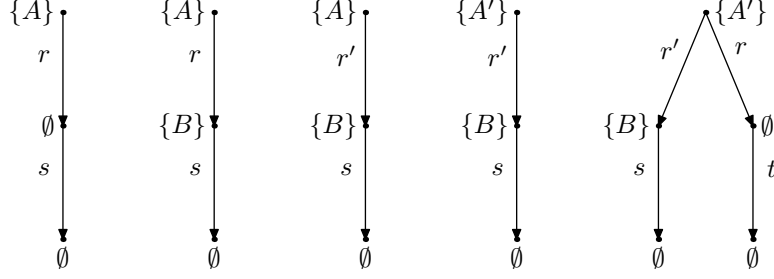


Figure 4.3: The tree on the left is refined stepwise to the tree on the right, where we assume a knowledge base $\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$. The operator performs four different kinds of operations (from left to right): 1. label extension (B added), 2. edge refinement (r replaced by r'), 3. label refinement (A replaced by A'), 4. attaching a subtree ($\exists r. \exists t. \top$ added).

- $re(t, v)$: $t' \in re(t, v)$ iff $t' \in T_{min}$ and $t' = (V, E', \ell)$, where $E' = E \setminus \{(v, r, w)\} \cup \{(v, r', w)\}$ for some $(v, r, w) \in E$ and $r' \in sh_{\downarrow}(r)$

Algorithm 2: Computation of the set $as(t, v)$

```

1  $\mathcal{T} := \emptyset$ ;  $\mathcal{M} := \{(t_{\top}, N_R)\}$ ;
2 while  $\mathcal{M} \neq \emptyset$  do
3   choose and remove  $(t', \mathcal{R}) \in \mathcal{M}$ ;
4    $\mathcal{R}' := \max(\mathcal{R})$ ;  $\mathcal{R}'' := \emptyset$ ;
5   while  $\mathcal{R}' \neq \emptyset$  do
6     choose and remove  $r \in \mathcal{R}'$ ;
7      $t'' := t[v \leftarrow (r, t')]$ ;  $w := \text{root}(t')$ ;
8     if  $t''$  is minimal then
9        $\mathcal{T} := \mathcal{T} \cup \{t''\}$ ;
10    else
11      forall  $(v, r', w') \in E(t'')$  with  $w \neq w'$  and  $r \sqsubseteq r'$  do
12        if  $t''(w) \preceq t''(w')$  then
13          nextwhile;
14       $\mathcal{R}' := \mathcal{R}' \cup (sh_{\downarrow}(r) \cap \mathcal{R})$ ;  $\mathcal{R}'' := \mathcal{R}'' \cup \{r\}$ ;
15     $\mathcal{M} := \mathcal{M} \cup \{(t^*, \mathcal{R}'') \mid t^* \in \rho(t'), \mathcal{R}'' \neq \emptyset\}$ ;
16 return  $\mathcal{T}$ ;
    
```

The crucial part of the refinement operator is the attach subtree operation, which is defined by Algorithm 2. The set $as(t, v)$ consists of minimal trees obtained from t that have an extra subtree attached to v . It recursively calls the refinement operator ψ and we therefore give its definition before we explain $as(t, v)$ in more detail.

Definition 4.21 (\mathcal{EL} Refinement Operator ψ)

The refinement operator $\psi : T_{min} \rightarrow \mathcal{P}(T_{min})$ is defined as:

$$\psi(t) := \bigcup_{v \in V(t)} (el(t, v) \cup rl(t, v) \cup re(t, v) \cup as(t, v))$$

□

For $t \in T_{min}$ and $v \in V$, Algorithm 2 keeps a set of output trees \mathcal{T} and a set \mathcal{M} of candidates which are tuples consisting of a minimal \mathcal{EL} tree and a set of role names. Within the first while loop, an element (t', \mathcal{R}) is removed from \mathcal{M} . The set \mathcal{R}' is initialized to contain the greatest elements of \mathcal{R} , and \mathcal{R}'' is initially empty and will later contain role names that need further inspection. In the second while loop, the algorithm iterates over all role names r in \mathcal{R}' . First, the tree t'' is constructed from t by attaching the subtree (v, r, w) to v , where w is the root of t' . It is then checked whether t'' is minimal. If this is the case, t'' is a refinement of t and is added to \mathcal{T} . Otherwise there are two reasons why t'' is not minimal: Either the newly attached subtree is subsumed by some other subtree of t , or the newly attached subtree subsumes some other subtree of t . The latter case is checked in Line 11, and if it applies the algorithm skips the loop. This prevents the algorithm from running into an infinite loop, since we would not be able to refine t' until t'' becomes a minimal tree. Otherwise in the former case, we proceed in two directions. First, $sh_{\downarrow}(r)$ is added to \mathcal{R}' , so it can be checked in the next round of the second while loop whether t' attached via some $r' \in sh_{\downarrow}(r) \cap \mathcal{R}$ to v yields a refinement. Second, we add r to \mathcal{R}'' , which can be seen as “remembering” that r did not yield a refinement in connection with t' . Finally, once \mathcal{R}' is empty, in Line 19 we add all tuples (t^*, \mathcal{R}'') to \mathcal{M} , where t^* is obtained by recursively calling ψ on t' .

Example 4.22 (Application of ψ)

Let \mathcal{K} be the following knowledge base:

$$\begin{aligned} N_C &= \{\text{Human}, \text{Animal}, \text{Bird}, \text{Cat}\} \\ N_R &= \{\text{has}, \text{has_child}, \text{has_pet}\} \\ \mathcal{K} &= \{\text{has_pet} \sqsubseteq \text{has}, \text{has_child} \sqsubseteq \text{has}, \text{Bird} \sqsubseteq \text{Animal}, \text{Cat} \sqsubseteq \text{Animal}\} \end{aligned}$$

Figure 4.4 depicts the set of all trees in $\psi(\text{Human} \sqcap \exists \text{has}.\text{Animal})$ w.r.t. \mathcal{K}' . □

Proposition 4.23 (Finiteness, Properness, Weak Completeness of ψ)

ψ is a finite, proper and weakly complete downward refinement operator on (T_{min}, \preceq) .

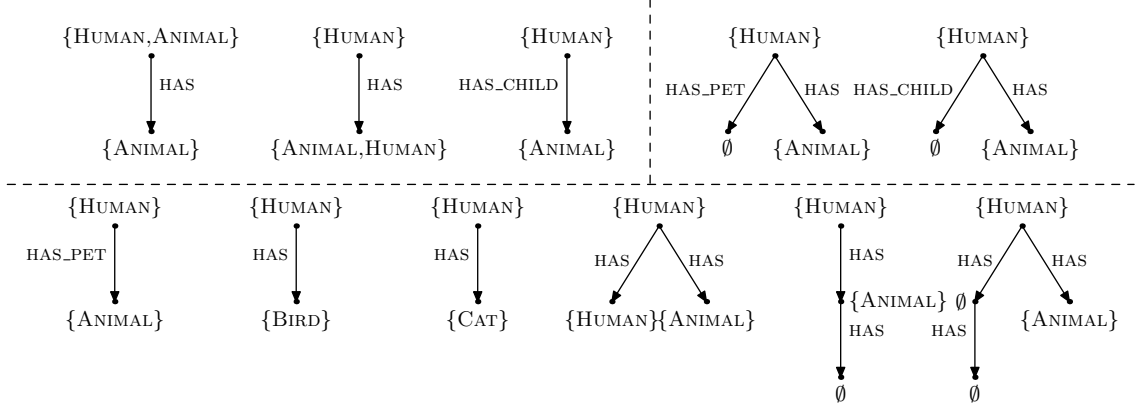


Figure 4.4: The set $\psi(\text{Human} \sqcap \exists \text{has. Animal})$ of minimal trees w.r.t. the knowledge base \mathcal{K}' from Example 4.22.

PROOF In the following, let $t \in T_{\min}$ and $v \in V(t)$.

First, it is easily seen that ψ is a downward refinement operator. Every operation of ψ adds a label or a subtree to a node v , or replaces a label or edge-label by a refined label or edge-label respectively. Hence, $t' \preceq t$ for all $t' \in \psi(t)$.

Regarding finiteness of ψ , Lemma 4.19 guarantees that there is only a finite number of minimal \mathcal{EL} trees up to a fixed depth. It then follows from Lemma 4.20 that for a given tree t , $\psi(t)$ only consists of trees of depth at most $\text{depth}(t) + 1$. Hence, $\psi(t)$ is finite.

In order to prove properness of ψ , it is sufficient to show $t \not\preceq t'$ for $t' \in \psi(t)$. To the contrary, assume $t \preceq t'$ and that t has been refined at v . Let \mathcal{S} be a simulation from t' to t . Since v has been refined, it follows that $(v, v) \notin \mathcal{S}$. We have that \mathcal{S} is a simulation, so there must be some $v' \in V(t)$ with $\text{level}(v') = \text{level}(v)$ such that $(v', v) \in \mathcal{S}$. This implies that there is a simulation \mathcal{S}' on t' with $\{(v', v), (v, v)\} \subseteq \mathcal{S}'$. It follows that t' contains a redundant subtree at the predecessor of v , contradicting to the minimality of t' .

Regarding weakly completeness, let $\text{depth}(t) \leq n$. We show that t is reachable from t_{\top} by nested induction on n and $m := |\{(root(t), r, w) \in E(t)\}|$. For the induction base case $n = 0, m = 0$, t is just a single node labeled with some concept names. It is easily seen that by repeatedly applying $el(t, v)$ and $rl(t, v)$ to this node we eventually reach t . For the induction step, let $n > 0, m > 0$. Hence, the root of t has m successor nodes w_1, \dots, w_m attached along edges r_1, \dots, r_m to t . By the induction hypothesis, the tree t_{m-1} , which is obtained from t by removing the subtree $t(w_1)$ from t , is reachable from t_{\top} . Also, there is a refinement chain θ from t_{\top} to $t(w_1)$ such that an intermediate tree t'_{w_1} occurs in θ and $t' = t_{m-1}[root(t) \leftarrow (r'_1, t'_{w_1})] \in as(t_{m-1}, root(t))$ for some r'_1 with $r_1 \sqsubseteq r'_1$. Hence, we can first reach t' from t_{\top} and then, by applying the remaining refinement steps from θ to t' and refining r'_1 to r_1 , eventually reach t . ■

Still, ψ is not ideal, since it is not complete. It is however easy to derive a

complete operator ψ^* from ψ :

$$\psi^*(t) := \max\{t' \mid t_{\top} \rightsquigarrow_{\psi}^* t', t' \prec t \text{ and } \text{depth}(t') \leq \text{depth}(t) + 1\}.$$

This construction is needed, because we would for example not be able to reach $\exists r.(A_1 \sqcap A_2)$ starting from $\exists r.A_1 \sqcap \exists r.A_2$ with ψ .

Theorem 4.24 (Ideality of ψ^*)

The \mathcal{EL} downward refinement operator ψ^ is ideal.*

Previously, we have shown that for languages other than \mathcal{EL} complete and non-redundant refinement operators do not exist (under a mild assumption). The same result carries over to our setting:

Proposition 4.25 (No Redundant and Complete \mathcal{EL} Operators)

Let $\psi : T_{\min} \rightarrow \mathcal{P}(T_{\min})$ be a complete refinement operator. Then ψ is redundant.

PROOF We assume $\mathcal{K} = \emptyset$ and N_C contains A_1 and A_2 . Since ψ is complete and its refinements are minimal, we have $\top \rightsquigarrow^* A_1$. Similarly, $\top \rightsquigarrow^* A_1$, $A_1 \rightsquigarrow^* A_1 \sqcap A_2$, and $A_2 \rightsquigarrow^* A_1 \sqcap A_2$. We have $A_1 \not\sqsubseteq A_2$ and $A_2 \not\sqsubseteq A_1$, which means that $A_1 \not\rightsquigarrow^* A_2$ and $A_2 \not\rightsquigarrow^* A_1$. Hence, $A_1 \sqcap A_2$ can be reached from \top via a refinement chain going through A_1 and a different refinement chain not going through A_1 , i.e. ψ is redundant. ■

Optimisations

We used two different kinds of optimisations: The first is concerned with the performance of minimality tests and the second reduces the number of trees returned by ψ by incorporating more background knowledge.

Recall from Section 4.2.1 that checking for minimality of a tree t involves computing a maximal simulation \mathcal{S} on $V(t)$ and is in $\mathcal{O}(|t|^2)$. In order to avoid expensive re-computations of \mathcal{S} after each refinement step, the data-structure of t is extended such that sets $\mathcal{C}_1^-(v)$, $\mathcal{C}_1^+(v)$, $\mathcal{C}_2^-(v)$ and $\mathcal{C}_2^+(v)$ are attached to every node $v \in V(t)$. Here, the set $\mathcal{C}_1^-(v)$ contains those nodes w such that (SC1) holds for (v, w) according to Definition 4.11. Likewise, $\mathcal{C}_2^+(v)$ is the set of those nodes w such that (SC2) holds for (w, v) , and $\mathcal{C}_1^+(v)$ and $\mathcal{C}_2^-(w)$ are defined accordingly. When checking for minimality, it is moreover sufficient that each such set

is restricted to only consist of nodes from $onlevel(v)$ excluding v itself. This fragmentation of \mathcal{S} allows us to perform local updates instead of re-computation of \mathcal{S} after an operation is performed on v . For example, when the label of v is extended, we only need to recompute $\mathcal{C}_1^-(v)$, update $\mathcal{C}_1^-(w)$ for every $w \in \mathcal{C}_1^-(v)$, and then repeatedly update $\mathcal{C}_2^-(v')$ and $\mathcal{C}_2^-(v')$ for every predecessor node v' of an updated node until we reach the root of t . For each of the operations (refining/extending label, refining/adding edge) performed by ψ , we developed a local simulation update algorithm. This method saves a considerable amount of computation, since the number of nodes affected by an operation is empirically relatively small.

In order to keep the number of refinements $|\psi(t)|$ small, we use role domains and ranges as well as disjoint concepts inferred from \mathcal{K} . This is similar to what we did for ρ in Section 4.1. We therefore refrain from spelling out the formal details. The domain restriction axioms can be used to reduce the set of role names considered when adding a subtree or refining an edge: For instance, let w be a node, (v, r, w) the edge pointing to w , and $range(r) = A$. When adding an edge (w, s, u) , we verify that $range(r) \sqcap domain(s)$ is satisfiable. This ensures that only compatible roles are combined. In ρ , we achieved this by restricting concepts of the form $\exists r.\top$ to those where $r \in mgr_B$ (see page 61ff). Similar optimisations can be applied to edge refinement. In $as(t, v)$, we furthermore use range restrictions to automatically label a new node with the corresponding role range. For example, if the edge has label r and $range(r) = A$, then the new node w is assigned label $\ell(w) = \{A\}$ (instead of $\ell(w) = \emptyset$).

We addressed the optimisation of extending node labels in the implementation of the function $e\ell$. Let A be a concept name for which we want to know whether or not we can add it to $\ell(v)$. We first check $A \sqsubseteq \ell(v)$. If yes, we discard A since we could reach an equivalent concept by refining a concept in $\ell(v)$, i.e. we perform redundancy reduction. Let (u, r, v) be the edge pointing to v and $range(r) = B$. We verify that $A \sqcap B$ is satisfiable and discard A otherwise. Additionally as before, we test whether $\ell(v) \sqsubseteq A$. If yes, then A is also discarded, because adding it would not result in a proper refinement. Performing the last step in a top down manner, i.e. start with the most general concepts A in the class hierarchy, ensures that we compute the maximum of eligible concepts, which can be added to $\ell(v)$. In summary, we make sure that the tree we obtain is label reduced, and perform an on-the-fly test for its satisfiability. Applying similar ideas to the case of label refinement is straightforward.

In practice, the techniques briefly described in this section narrow the set of trees returned in a refinement step significantly by ruling out concepts, which are unsatisfiable w.r.t. \mathcal{K} or which can also be reached via other refinement chains. This is illustrated by the following example.

Example 4.26 (Application of Extended Operator ψ)

Let \mathcal{K} be as in Example 4.22 and define $\mathcal{K}' := \mathcal{K} \cup \{domain(has_child) = Human, range(has_child) = Human, domain(has_pet) = Human, range(has_pet) = Animal, Human \sqcap Animal \equiv \perp\}$. By incorporating the additional axioms, $\psi(Human \sqcap$

$\exists \text{has.Animal}$) contains less refinements. In Figure 4.22, the trees on the top left are no refinements with respect to the more expressive knowledge base. For the first two trees, this is due to the disjointness of **Human** and **Animal**. The third tree is ruled out, since the range of **hasChild** is incompatible with **Animal**. The trees on the top right are modified: \emptyset is replaced by **Animal** and **Human**, respectively, due to the ranges of **has_child** and **has_pet**. \square

4.2.3 Operator Performance

In order to evaluate the operator, we computed *random refinement chains* of ψ . A random refinement chain is obtained by applying ψ to \top , choosing one of the refinements uniformly at random, then applying ψ to this refinement, etc.

Name	Logical axioms	Classes	Roles	ψ av. time (in ms)	ψ per ref. (in ms)	Reasoning time (%)	Refinements (av. and max.)		Ref. size (av. and max.)	
GENES	42656	26225	4	167.2	0.14	68.4	1161.5	2317	5.0	8
CTON	33203	17033	43	76.2	0.08	5.1	220.2	28761	5.8	24
GALEN	4940	2748	413	3.5	0.21	37.1	17.0	346	4.9	16
PROCESS	2578	1537	102	193.6	0.16	27.2	986.5	23012	5.7	22
TRANSP.	1157	445	89	164.4	0.09	5.9	985.2	22651	5.7	24
EARTH.	931	559	81	407.4	0.17	23.2	1710.3	27163	5.7	19
TAMBIS	595	395	100	141.6	0.09	1.5	642.4	26685	5.8	23

Table 4.4: Benchmark results on ontologies from the TONES repository. The results show that ψ works well even on large knowledge bases. The time needed to compute a refinement is below one millisecond and does not show large variations.

To assess the performance of the operator, we tested it on real ontologies chosen from the TONES repository², including some of the most complex OWL ontologies. We generated 100 random refinement chains of length 8 and measured the results. We found experimentally that this allows us to evaluate the refinement operator on a diverse set of concept trees. The tests were run on an Athlon XP 4200+ (dual core 2.2 GHz) with 4 GB RAM. As a reasoner we used Pellet 1.5. The benchmarks do not include the time to load the ontology into the reasoner and classify it.

The results are shown in Table 4.4. The first four columns contain the name and relevant statistics of the ontology considered. The next column shows the average

²<http://owl.cs.manchester.ac.uk/repository/>

time the operator needed on each input concept. In the following column this value is divided by the number of refinements of the input concept. The subsequent column shows how much time is spend on reasoning during the computation of refinements. The two last columns contain the number of refinements obtained and their size. Here, we measure size as the number of nodes in a concept tree plus the sum of the cardinality of all node labels.

The most interesting insight from Table 4.4 is that despite the different size and complexity of the ontologies, the time needed to compute a refinement is low and does not show large variations (between 0.09 and 0.21 ms). This indicates that the operator scales well to large knowledge bases. It can also be observed that the number of refinements can be very high in certain cases, which is due to the large number of classes and properties in many ontologies and the absence of explicit or implicit disjointness between classes. We want to note that when the operator is used to learn concepts from instances (standard learning task), one can use the optimisations in Section 4.2.2 and consider classes without common instances instead of class disjointness. In this case, the number of refinements of a given concept will usually be much lower, since no explicit disjointness axioms are required. In all experiments we also discovered that the time the reasoner requires differs a lot (from 1.5% to 68.4%). However, since the number of reasoner requests is finite and the results are cached, this ratio will decrease with more calls to the refinement operator. Summing up, the results show that efficient ideal refinement on large ontologies can be achieved in \mathcal{EL} , which in turn is promising for \mathcal{EL} concept learning algorithms. In future work, we want to investigate whether certain extensions of \mathcal{EL} may be supported by the operator without losing ideality.

In summary, we have provided an efficient ideal \mathcal{EL} refinement operator, thereby closing a gap in refinement operator research. We have shown that the operator can be applied to very large ontologies and makes profound use of background knowledge. In Section 5.2, we will incorporate the ψ refinement operator in the ELTL learning algorithm.

5 Refinement Operator Based OWL Learning Algorithms

In this chapter, we describe how to construct a learning algorithm given a refinement operator and a learning problem. We will introduce three algorithms with different strengths: OCEL (OWL Class Expression Learner), ELTL (EL Tree Learner), and CELOE (Class Expression Learning for Ontology Engineering). OCEL is the standard learning algorithm using the presented OWL refinement operator ρ . ELTL is an algorithm optimised for learning EL trees using the EL refinement operator ψ and CELOE is a variant of OCEL, which contains adaptations specific for the class learning problem.

A learning algorithm can be constructed as a combination of a refinement operator, which defines how the search tree can be built, and a search algorithm, which controls how the tree is traversed. This is illustrated in Figure 5.1. Assuming a top-down approach is used, an algorithm could start with the top concept, apply the operator to obtain refinements, e.g. **Person**, **Car**, and **Building**. It can then evaluate those concepts. Using a heuristic search, the best node is expanded in the next step. Expanding the node involves applying the operator again, evaluating all refinements etc. When a termination criterion is satisfied, i.e. a concept in the search tree is sufficiently good with respect to the learning problem, the algorithm stops and outputs a solution. All three algorithms in this chapter follow this basic principle with some deviations, which will be described.

5.1 OCEL (OWL Class Expression Learner)

In Section 4.1, we have designed a complete and proper operator ρ^{cl} . Unfortunately, such an operator has to be redundant and infinite by Theorem 3.12. We will now describe how to deal with these problems and define a learning algorithm.

5.1.1 Redundancy Elimination

Whenever a learning algorithm encounters a node in the search tree, it could check whether a weakly equal concept already exists in the tree. If yes, then this node is ignored, i.e. it will not be expanded further and it will not be evaluated. This removes all redundancies, since every concept exists at most once in the search

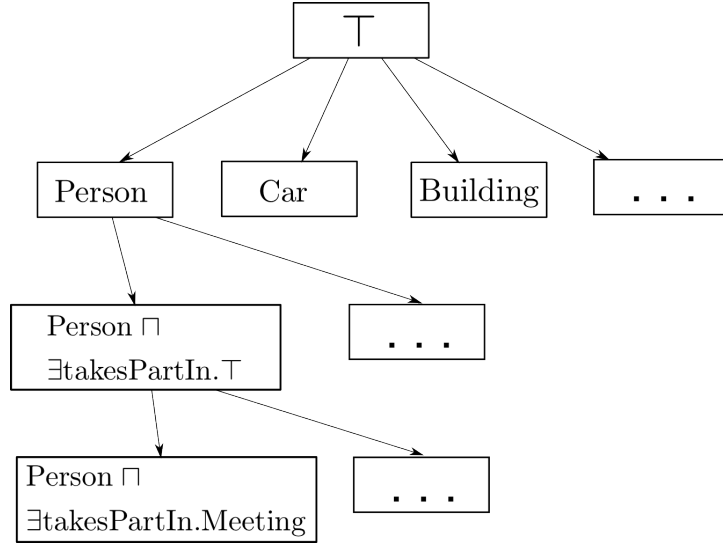


Figure 5.1: Illustration of a search tree in a top down refinement approach.

tree.¹ We can still reach all concepts, because we have $\rho^{cl}(C) \simeq \rho^{cl}(D)$ if $C \simeq D$, i.e. ρ^{cl} handles weakly equal concepts in the same way. However, this redundancy elimination approach is computationally expensive if performed naively. Hence, we considered it worthwhile to investigate how it can be handled efficiently.

Note, that we consider weak equality instead of equality here, e.g. we have $A_1 \sqcap A_2 \neq A_2 \sqcap A_1$, but $A_1 \sqcap A_2 \simeq A_2 \sqcap A_1$. We do this, because having $A_1 \sqcap A_2$ and $A_2 \sqcap A_1$ – while not being syntactically equal – can still be considered redundant and should be avoided. In conjunctions and disjunctions, this raises the problem that we have to guess which pairs of elements are equal to determine whether two concepts are weakly equal. One way to solve this problem is to define an ordering over concepts and require the elements of disjunctions and conjunctions to be ordered accordingly. This eliminates the guessing step and allows to check weak equality in linear time. There are different ways to define a linear order \preceq over concepts. It is also possible to do it a way so that deciding \preceq for two concepts is polynomial and transforming a concept in negation normal form to \preceq *ordered negation normal form*, i.e. elements in conjunctions and disjunctions are ordered with respect to \preceq , can be done in polynomial time – we omit the straightforward details. It is thus reasonable to assume that every concept occurring in the search tree can be transformed to ordered negation normal form with respect to some linear order over concepts. We can then maintain an ordered set of concepts occurring in the search tree. Checking weak equality of a concept C with respect to a search tree containing n concepts will then only require $\log n$ comparisons (binary search), where each comparison needs only linear time. Compared to an algorithm without redundancy check, this can avoid many concept tests. Each

¹More precisely: For each concept there is at most one representative of the equivalence class of weakly equal concepts in the search tree which has been evaluated.

concept tests requires potentially expensive instance checks. The complexity of instance checks is EXPTIME for \mathcal{ALC} , NEXPTIME for $\mathcal{SHOIN}(D)$ and OWL-DL, and 2NEXPTIME for $\mathcal{SROIQ}(D)$ and OWL 2 DL. Taking this into account, redundancy elimination can be considered reasonable. Indeed, in our experimental evaluation, redundancy checks typically require only one percent of the overall algorithm runtime.

5.1.2 Creating a Full Learning Algorithm

Learning concepts in DLs is a search process. In the OCEL learning algorithm, the refinement operator ρ^{cl} (see page 70) is used for building the search tree, while the heuristic in Definition 5.1 decides which nodes to expand. As mentioned in Section 4.1.3, we want to tackle the infinity of the operator by considering only refinements up to some length n at a given time. We call n the *horizontal expansion* of a node in the search tree. It is a node specific upper bound on the length of child concepts, which can be increased dynamically by the algorithm during the learning process.

To deal with this, we formally define a *node* in a search tree to be a triple (C, n, b) , where C is a concept, $n \in \mathbb{N}$ is the horizontal expansion, and $b \in \{\text{true}, \text{false}\}$ is a boolean marker for the redundancy of a node.

To define a search heuristic for our learning algorithm, we need some notions to be able to express what we consider a good node for expansion. Similarly to existing ILP systems, we use the learning algorithm parameter *noise*, bounding the minimum acceptable training set accuracy of the learned definition. $(1 - \text{noise})$ is the lowest accuracy a concept needs to have to be considered a solution of a learning problem.

The search heuristic selects the node with the highest score in the search tree at a given time, where the score of a node is defined as follows:

Definition 5.1 (OCEL Score)

Let $N = (C, n, b)$ be a node. We introduce the following notions:

$$\begin{aligned} \text{accuracy}(C) &= 1 - \frac{up + cn}{|E|} \\ \text{acc_gain}(N) &= \text{accuracy}(C) - \text{accuracy}(C') \\ &\quad \text{where } C' \text{ is the concept in the parent of } N \\ up &= |E^+ \setminus R(C)| \quad (\text{uncovered positives}) \\ cn &= |R(C) \cap E^-| \quad (\text{covered negatives}) \end{aligned}$$

If $up > \lfloor \text{noise} \cdot |E| \rfloor$, then the node is *too weak* in this noise setting, i.e. it is not a solution candidate and will never be expanded. If the node is not too weak, then its score is defined as follows:

$$\text{score}(N) = \text{accuracy}(C) + \alpha \cdot \text{acc_gain}(N) - \beta \cdot n \quad (\alpha \geq 0, \beta > 0) \quad \square$$

By default, we choose $\alpha = 0.5$ and $\beta = 0.02$. The heuristic uses predictive accuracy as main criterion. Accuracy gain, controlled by α , is incorporated, because those concepts having lead to an improvement in accuracy are more likely to be significant refinements towards a solution. As a third criterion, controlled by β , we bias the search towards shorter concepts and less explored areas of the search space. Using horizontal expansion instead of concept length as factor makes the algorithm more flexible in searching less explored areas of the search space and avoids that it gets stuck on concepts with high accuracy and accuracy gain. The score function can be defined independently of the core learning algorithm, i.e. we can easily replace it with a different one. For instance, for some problems one type of error is more severe than another type, e.g. not detecting cancer is worse than erroneously detecting cancer. This can be modelled by assigning different weights to *up* and *cn* in Definition 5.1.

We have now introduced all necessary notions to specify the complete learning algorithm, given in Algorithm 3. *checkRed* is the redundancy check function and *transform* the function to transform a concept to ordered negation normal form as described in Section 5.1.1.

Algorithm 3: OCEL Learning Algorithm

Input: background knowledge, examples E , *noise* in $[0,1]$

- 1 ST (search tree) is set to the tree consisting only of the root node $(\top, 0, false)$
- 2 **while** ST does not contain a node with $q < \lfloor noise \cdot |E| \rfloor$ **do**
- 3 choose a node $N = (C, n, b)$ with highest score in ST
- 4 expand N up to length $n + 1$, i.e. :
- 5 **begin**
- 6 add all nodes $(D, n, checkRed(ST, D))$ with $D \in transform(\rho^{cl}(C))$
and $|D| = n + 1$ as children of N
- 7 evaluate created non-redundant nodes
- 8 change N to $(C, n + 1, b)$
- 9 **end**
- 10 Return found concepts in ST

As we can see, the learning algorithm performs a top down refinement operator driven heuristic search. The main difference to other learning algorithms of this kind is the replacement of a full node expansion by a one step horizontal expansion and the use of a redundancy check procedure.

Apart from knowledge representation, another difference to many ILP programs is the search strategy: Often, ILP tools perform a clause by clause set covering approach to construct a solution stepwise (see also Section 2.2.1). In OCEL, each concept represents a full solution, which is related to single predicate theory learning [Bratko, 1999] in ILP. A benefit is that there is no risk in performing possibly suboptimal choices and it is often possible to learn shorter solutions. However,

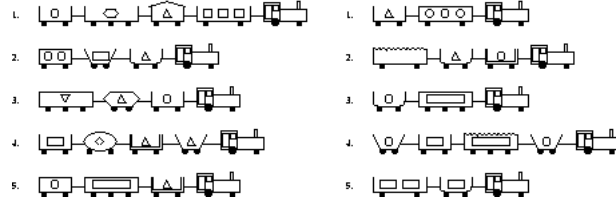


Figure 5.2: The Michalski trains problem: positive examples are on the left, negative examples are on the right.

it also leads to a higher runtime and memory consumption. To counteract this, a divide and conquer strategy as extension of Algorithm 3 can be activated in the DL-Learner implementation of OCEL. It restricts the set of nodes which are candidates for expansion to a set of fixed size in regular time intervals. By default, the candidate set is restricted to the 20 most promising nodes each 300 seconds. Those concepts are selected according to their accuracy with a bias towards short concepts with high accuracy on positive examples, since those concepts are more likely to improve in a downward refinement algorithm. We omit the details of this process for brevity. It can be used as a tradeoff between performance and the risk to make suboptimal decisions.

Correctness of the algorithm can be shown:

Proposition 5.2 (Correctness)

If a learning problem has a solution in \mathcal{ALC} , then Algorithm 3 terminates and computes a correct solution of the learning problem.

PROOF Assume, there is a solution C (which is an \mathcal{ALC} concept) of a learning problem. By the weak completeness of ρ^{cl} , we know that there is a concept D with $D \equiv C$ and $D \in \rho^*(\top)$, i.e. ρ^{cl} allows a refinement chain $\top \rightsquigarrow D_1 \rightsquigarrow D_2 \rightsquigarrow \dots \rightsquigarrow D_n = D$. We have already shown in Lemma 4.9 that ρ^{cl} does not reduce length, i.e. all concepts in this chain have at most the length of D . This means that the *score* of each node is higher than $-|D|$ where $|D|$ is the length of D . Because β in the score function is higher than 0, any node with sufficiently high horizontal expansion has a score lower than $-|D|$. As a consequence, all nodes in our chain will eventually be expanded sufficiently often to refine to its successor in the chain above, i.e. eventually D will be reached unless the algorithm terminates with a different solution beforehand. In both cases the proposition is satisfied. ■

Example 5.3 (OCEL)

We illustrate the OCEL algorithm using Michalski’s trains [Michalski, 1980] as a simple example. The data describes different features of trains, e.g. which cars

are appended to a train, whether they are short or long, closed or open, jagged or not, which shapes they contain and how many of them. The positive examples are the trains on the left in Figure 5.2 and the negative examples are the trains on the right. Thus, the task of the learner is to find characteristics of all the left trains, which none of the right trains has. The learning algorithm first explores the concepts \top and then **Train**, which cover all examples. Other atomic concepts are too weak to be considered for further exploration. The exploration of the top concept up to a horizontal expansion of 3 leads to $\exists \text{hasCar}.\top$, which is then expanded to $\exists \text{hasCar}.\text{Closed}$. This covers all positives and two negatives. The heuristic later picks this node and extends it up to a horizontal expansion of 5 to $\exists \text{hasCar}.\text{(Closed} \sqcap \text{Short)}$, which is a possible (and shortest) solution for the problem. \square

5.2 ELTL (EL Tree Learner)

ELTL (\mathcal{EL} Tree Learner) is a learning algorithm building on the introduced \mathcal{EL} refinement operator ψ (see Definition 4.21 on page 84). Since ψ is ideal, we can omit some of the constructs in the introduced OCEL algorithm (Algorithm 3).

In particular, we do not need a stepwise horizontal expansion. This technique was introduced to overcome the problem that ρ is not finite. It was implemented by attaching an integer value to each node in the search tree. In ELTL, we omit this value, i.e. a node in an ELTL search tree is defined as a tuple (C, b) where C is a concept and $b \in \{\text{true}, \text{false}\}$ is a boolean marker for the redundancy of a node. In an ELTL search tree, each node is expanded at most once, i.e. the candidates for expansion are the childless nodes in the search tree.

Algorithm 4 shows the basic ELTL algorithm, which is a simplified variant of OCEL. To complete the algorithm, we need to introduce a heuristic. To do this, we adapt the OCEL score in Definition 5.1 by replacing the horizontal expansion penalty with a penalty for the length of a concept. Otherwise, the same redundancy elimination strategy as in OCEL is used.

Algorithm 4: ELTL Base Learning Algorithm

Input: background knowledge, examples E , *noise* in $[0,1]$

- 1 ST (search tree) is set to the tree consisting only of the root node (\top, false)
- 2 **while** ST does not contain a node with $q < \lfloor \text{noise} \cdot |E| \rfloor$ **do**
- 3 choose a child-less node $N = (C, b)$ with highest score in ST
- 4 expand N , i.e. :
- 5 **begin**
- 6 add all nodes $(D, \text{checkRed}(ST, D))$ as children of N
- 7 evaluate created non-redundant nodes
- 8 **end**
- 9 Return found concepts in ST

Depending on the application scenario, \mathcal{EL} may not be expressive enough to solve a learning problem. In particular, it does not support disjunction of concepts. To be able to combine the benefit of the ideal operator ψ and the expressive power of disjunction, we can employ a covering approach. We refer to Section 2.2.1 for an overview over programs using such an approach. Multiple clauses in a logic program can be viewed as disjunction of clause bodies ($(r \rightarrow t) \wedge (s \rightarrow t)$ is equivalent to $(r \vee s) \rightarrow t$ in propositional logic). Therefore, the idea of the (disjunctive) ELTL algorithm is to reuse ILP methodology by learning several \mathcal{EL} trees and connecting them by disjunction.

Algorithm 5 shows the corresponding learning algorithm. It only returns a single concept C , whereas OCEL returns a set of most promising concepts. C is initialised as the bottom concept, i.e. does not cover any examples. In the inner **while** loop, an \mathcal{EL} tree, corresponding to a concept C' , is learned. This tree is added to the current solution disjunctively, i.e. C is set to $C \sqcup C'$. The examples are updated accordingly, i.e. positive and negative examples covered by C' are removed. The algorithm stops when the score of the last tree found in the inner **while** loop is below a specified threshold, which is given as parameter *minTreeScore*.

Algorithm 5: ELTL Learning Algorithm

Input: background knowledge base \mathcal{K} , pos. and neg. examples E^+ and E^- ,
noise in $[0,1]$, *secondsPerTree* $\in \mathcal{R}$, *minTreeScore* in $[0,1]$

- 1 initialise $C = \perp$ and *bestTreeScore* = 1
- 2 **while** *bestTreeScore* \geq *minTreeScore* **do**
- 3 ST is set to the tree consisting only of the root node (\top, false)
- 4 **while** *less than secondsPerTree seconds have elapsed and* ST *does not*
 contain a correct solution w.r.t \mathcal{K}, E^+, E^- **do**
- 5 choose a child-less node $N = (C, b)$ with highest score in ST
- 6 expand N , i.e. :
- 7 **begin**
- 8 add all nodes $(D, \text{checkRed}(ST, D))$ as children of N
- 9 evaluate created non-redundant nodes
- 10 **end**
- 11 select C' from ST with the highest score *bestTreeScore*
- 12 **if** *bestTreeScore* \geq *minTreeScore* **then**
- 13 $C = C \sqcup C'$
- 14 $E^+ = E^+ \setminus R_{\mathcal{K}}(C')$
- 15 $E^- = E^- \setminus R_{\mathcal{K}}(C')$
- 16 Return *simplify*(C)

Naturally, we also need a termination criterion for the inner while loop. Different criteria exist in the ILP community, for instance achieving a minimum accuracy,

a minimum number of positive examples covered (in a top-down approach), a minimum accuracy gain, etc. While it would be straightforward to integrate those in the ELTL algorithm, we introduced runtime as another criterion. This means that the algorithm can spend a specified amount of time in the inner loop and we pick the best tree obtained during this time span. The reasoning behind this is that we can never be certain to find a better tree unless the problem is already correctly solved, in which case we also terminate the loop. Instead of deciding for a potentially suboptimal tree, we can use as much computational power as was assigned to the learning task. The overall algorithm runtime is, thus, more predictable. Of course, a disadvantage is that the final result depends on the used computer. It might potentially vary even on the same computer depending on CPU usage of other tasks. Initial tests indicate that runtime is a suitable criterion, but we defer a detailed analysis of different criteria to future work.

Finally, we need to specify the score function of the ELTL algorithm. We use the score function of the ELTL base algorithm explained above with two modifications:

1. We introduce a weight, which stronger penalises covering negative examples, i.e. a hypothesis covering two positive and two examples has a lower score than a hypothesis covering one positive and one negative example. This is done to avoid covering too many negative examples. While positive examples may still be covered by the next \mathcal{EL} tree generated in the inner loop of Algorithm 5, the effect of covering a negative example cannot be undone. This technique is also used in other ILP programs. In our case, we initially set the weight to 1.2 by default and let it converge to 1.0 depending on the number of trees learned.
2. We require that at least a specified fraction, by default 5%, of positive examples need to be covered by an \mathcal{EL} tree. This biases the learning algorithm towards short solutions and reduces overfitting.

Before the algorithm returns the solution C , we use a simplification mechanism to remove redundant parts of the solution. The simplifier first performs straightforward concept rewrites ($\perp \sqcup C$ replaced by C , $\forall r. \top$ replaced by \top etc.). After that, it uses a reasoner to further simplify the concept. For instance, given a concept $D_1 \sqcap \dots \sqcap D_n$, it checks whether D_1 is a super class of any D_i ($2 \leq i \leq n$). If this is the case, then the corresponding D_i can be removed. The algorithm continues with D_2 , D_3 etc. in the same manner. Disjunction can be handled analogously. The concept is simplified by this method in the sense that we cannot remove elements from conjunctions or disjunctions while maintaining equivalence. It is particularly useful in the ELTL algorithm, since several of the learned \mathcal{EL} trees may share similarities, which are unified by the simplifier. The method is powerful, since it can also detect non-obvious simplification through the use of OWL reasoners. Efficiency is not a significant problem, since the method is called only once before the algorithm terminates.

5.3 CELOE (Class Expression Learner for Ontology Engineering)

The last of the three algorithms, which we present, is particularly designed for ontology engineering. More specifically, it solves the concept learning problem in Definition 2.23. Given a class A in a knowledge base, the aim is to describe A formally. In particular, we want to learn a concept C , such that we can suggest adding axioms of the form $A \equiv C$ or $A \sqsubseteq C$ to a knowledge engineer.

CELOE builds on the OCEL algorithm, i.e. uses the same approach as in Algorithm 3. The main modification is a different heuristic. While we initially experimented with OCEL for the ontology engineering scenario by using instances of A as positive examples and non-instances of A as negative examples, we discovered that some modifications of the algorithms improve its performance. At the same time, many of the powerful configuration options of OCEL (not described in detail here) are not necessary. We therefore designed the CELOE algorithm. In this section, we will first briefly describe basic changes to OCEL in the algorithm core and then discuss the heuristic we employed.

One particular feature of CELOE is an even stronger bias towards short concepts. As we will describe in Section 7.3, we developed plugins using CELOE in ontology editors. In the ontology creation and maintenance scenario, it is unlikely that very long concepts are used. In contrast to this, OCEL is used for tasks like detecting whether chemical compounds cause cancer (see Section 7.2), where describing such structures requires much more complex concepts. Consequently, we introduced a strong bias in the CELOE heuristic towards short concepts, which means that the algorithm is less likely to explore and find more complex concepts. As a benefit, the algorithm is almost guaranteed to find any suitable short expression. In fact, the algorithm can be queried at runtime whether it has evaluated or pruned concepts up to a certain length. This can be used as additional information for the knowledge engineer to eliminate doubts on whether a shorter more appropriate suggestion exists.

We also use the ELTL simplifier described above to increase the readability of suggestions: For instance, $\exists \text{hasLeader}.\top \sqcap \text{Capital}$ is simplified to **Capital** if the background knowledge allows to infer that a capital is a city and each city has a leader. In addition to the standard mechanism, we use a reasoner cache, i.e. each inference drawn during the simplification of expressions is cached. Experimentally, it turned out that this allows an efficient minimisation of suggestions and has only marginal influence on the performance of the algorithm after an initial warm-up phase of the cache.

Furthermore, we also make sure that the suggestions are not “redundant”. If one suggestion is longer and subsumed by another suggestion and both have the same characteristics, i.e. classify the relevant individuals equally, the more specific suggestions are filtered. This avoids expressions containing irrelevant subexpressions and ensures that the suggestions are diverse.

In contrast to OCEL, where the noise parameter is used for deciding termination, we use a fixed runtime as parameter. This results in a more predictable behaviour for the knowledge engineer. In user interfaces, it allows to display a progress bar. Fixed runtime is also suitable, since the knowledge engineer in the ontology engineering case cannot be assumed to know or find good parameter settings. In fact, none of the options of CELOE are mandatory, which means that the user can directly query for suggestions without having to know internals of the algorithm or prepare anything.

We will describe the CELOE heuristic in the sequel. A heuristic measures how well a given class expression fits a learning problem and is used to guide the search in a learning process. To define a suitable heuristic, we first need to address the question of how to measure the accuracy of a class expression. We first introduce a straightforward way to achieve this and then describe the approach we take in CELOE.

As mentioned previously, we cannot simply use supervised learning from examples, since we do not have positive and negative examples available. We can straightforwardly tackle this problem by using the existing instances of the class as positive examples and the remaining instances as negative examples. This is illustrated in Figure 5.3, where \mathcal{K} stands for the knowledge base and A for the class to describe. We can then measure accuracy as the number of correctly classified examples divided by the number of all examples. This can be computed as follows for a class expression C :

$$acc_s(C) = 1 - \frac{|R(A) \setminus R(C)| + |R(C) \setminus R(A)|}{n} \quad n = |N_I|$$

$R(A) \setminus R(C)$ are the false negatives whereas $R(C) \setminus R(A)$ are false positives. n is the number of all examples, which is equal to the number of individuals in the knowledge base in this case. Apart from learning definitions, we also want to be able to learn super class axioms ($A \sqsubseteq C$). Naturally, in this scenario $R(C)$ should be a super set of $R(A)$. However, we still do want $R(C)$ to be as small as possible, otherwise \top would always be a solution. To reflect this in our accuracy computation, we penalise false negatives harder than false positives by a factor of t (where t should be greater than 1) and map the result in the interval $[0, 1]$:

$$acc_s(C, t) = 1 - 2 \cdot \frac{t \cdot |R(A) \setminus R(C)| + |R(C) \setminus R(A)|}{(t + 1) \cdot n} \quad n = |N_I|$$

While being straightforward, the outlined approach of casting class learning into a standard learning problem with positive and negative examples has the disadvantage that the number of negative examples will usually be much higher than the number of positive examples. As shown in Table 5.1, this may lead to overly optimistic estimates. More importantly, this accuracy measure has the drawback of having a dependency on the number of instances in the knowledge base. In order to overcome this problem, it is more appropriate to choose an approach

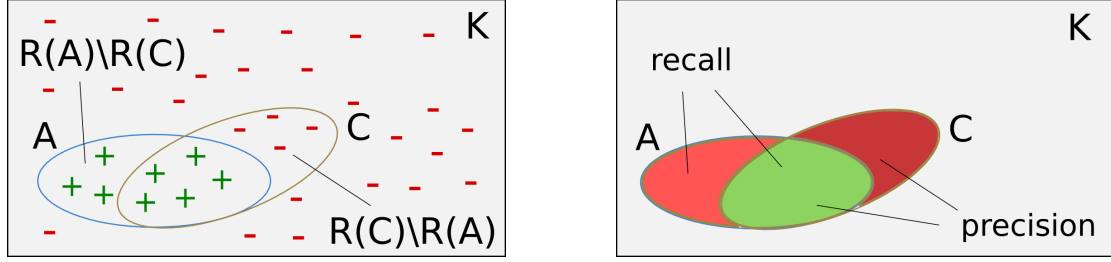


Figure 5.3: Visualisation of different accuracy measurement approaches. \mathcal{K} is the knowledge base, A the class to describe and C a class expression to be tested. Left side: Standard supervised approach based on using positive (instances of A) and negative (remaining instances) examples. Here, the accuracy of C depends on the number of individuals in the knowledge base. Right side: Evaluation based on two criteria: recall (Which fraction of A is covered by C ?) and precision (Which fraction of C is in A ?).

that avoids explicit usage of negative examples. We can view the problem from an information retrieval perspective, where $R(A)$ is the set of relevant objects and $R(C)$ the set of retrieved objects. We then combine precision and recall² as illustrated in Figure 5.3.

$$acc_c(C) = \frac{1}{2} \cdot \left(\frac{|R(A) \cap R(C)|}{|R(A)|} + \sqrt{\frac{|R(A) \cap R(C)|}{|R(C)|}} \right)$$

This is similar to F measure in information retrieval. The square root is used to better distinguish between large values of $|R(C)|$ (e.g. 1/100 differs only slightly from 1/1000), but can also be omitted. Again, we can use a factor t , enabling us to give the second criterion (precision) lower importance:

$$acc_c(C, t) = \frac{1}{t+1} \cdot \left(t \cdot \frac{|R(A) \cap R(C)|}{|R(A)|} + \sqrt{\frac{|R(A) \cap R(C)|}{|R(C)|}} \right) \quad (5.1)$$

Table 5.1 provides some example calculations and shows that this accuracy estimate is closer to intuition for the ontology engineering use case.

Computing accuracy serves two purposes: The first one is to present it to the knowledge engineer so that she can easily assess how good the expression fits the existing data. This is shown in Sections 7.3.1 and 7.3.2, which describe the ontology editor plugins based on the CELOE algorithm. The second purpose is to guide the learning algorithm at runtime towards a solution of the learning

²Precision is defined as the number of relevant documents retrieved by a search divided by the total number of documents retrieved by that search. Recall is defined as the number of relevant documents retrieved by a search divided by the total number of existing relevant documents.

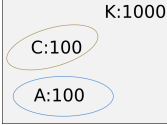
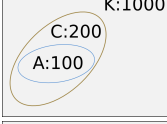
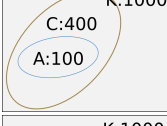
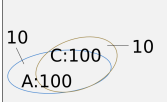
illustration	acc_s		acc_c	
	equivalence	super class	equivalence	super class
	80%	80%	0%	0%
	90%	95%	75%	88%
	70%	85%	63%	82%
	98%	98%	90%	90%

Table 5.1: Example accuracies for selected cases using $t = 3$ for the super class columns. The images on the left represent an imaginary knowledge base \mathcal{K} with 1000 individuals, where we want to describe the class A by using expression C . It is apparent that using predictive accuracy leads to impractical accuracies, e.g. in the first row C cannot possibly be a good description of A , but we still get 80% accuracy, since all the negative examples outside of A and C are correctly classified.

problem. In order to do this, we define the following notion of a score based on accuracy acc_c .

Definition 5.4 (CELOE Score)

Let A be the class to describe, C the expression to evaluate, and C' the parent of C in the search tree (i.e. C is a refinement of C'):

$$\begin{aligned}
 acc_gain(C) &= acc_c(C, t) - acc_c(C', t) \\
 score(C) &= acc_c(C, t) + \alpha \cdot acc_gain(N) - \beta \cdot |C| \quad (\alpha, \beta \geq 0) \quad \square
 \end{aligned}$$

The heuristic is composed of several criteria, the influence of which can be controlled by using α and β :

- accuracy ($acc_c(C, t)$): The main criterion as outlined above.
- accuracy gain ($\alpha \cdot acc_gain(N)$): If an expression has been refined to an expression with higher accuracy, it is more likely to be closer to a solution.
- length penalty ($\beta \cdot |C|$): Longer expressions are less readable for the knowledge engineer, so we penalise long expressions.

Typical values are $\alpha = 0.3$ and $\beta = 0.05$. For class learning β is set to a high value, which biases the search process towards short and, therefore, more readable solutions. The score is similar to the ones for OCEL and ELTL apart from the different accuracy estimate.

Now that we derived a heuristic, we should observe whether it is efficient to compute the score value in Definition 5.4. Almost all the time required to compute the score of a given expression is spent in calculating its accuracy. According to the definition of acc_c , we need to perform a retrieval of C . Performing such a retrieval can be very expensive for large knowledge bases. Depending on the ontology schema, this may require instance checks for many or even all objects in the knowledge base. An optimisation in order to compute the score efficiently, is to reduce the number of objects we are looking at by using background knowledge. Assuming that we want to learn an equivalence axiom for class A with super class A' , we can start a top-down search in our learning algorithm with A' instead of \top . Thus, we know that each expression we test is a subclass of A' , which allows us to restrict the retrieval operation to instances of A' . This can be generalised to several super classes and a similar observation applies to learning super class axioms. Chapter 6 introduces several further optimisations, which allow the score to be computed efficiently even if A' has many instances.

In summary, we defined three learning algorithms: OCEL, ELTL, and CELOE. OCEL is suitable for most standard machine learnings tasks. ELTL is designed to be very efficient and can solve tasks requiring only constructs in the \mathcal{EL} description logics – optionally augmented with disjunction. CELOE is similar to OCEL with several smaller algorithm adaptations and a different heuristic particularly designed for ontology engineering.

6 Improving Scalability of OWL Learning Algorithms

Performance and scalability are crucial factors in machine learning. This is particularly relevant for concept learning in description logics, which draws on complex inference mechanisms. In this chapter, we present optimisations of the algorithms introduced earlier to handle very large knowledge bases.

In the first part, we present DBpedia, which is a project maintaining a very large knowledge base extracted from Wikipedia.¹ DBpedia is a typical example of a large knowledge base in the Linking Open Data (LOD) cloud. It also served as test bed for the scalability evaluation of learning algorithms. Note that the description of DBpedia is not strictly necessary for understanding other parts of the thesis and can be skipped by the reader.

In the second part, we present an approach, which allows to extract fragments of a knowledge base such that learning on the fragment yields similar results like learning on the whole knowledge base, but is more efficient. This is a prerequisite for the learning algorithms to work on very large knowledge bases, since some of those knowledge bases including DBpedia cannot be handled efficiently by reasoners. Additionally, this allows the algorithms to be applied in scenarios where only parts of a knowledge base are consistent.

The third part of the chapter is concerned with optimising the performance of the coverage test, which is necessary to compute the heuristic function in learning algorithms. Since most of the time of an algorithm run is required by coverage tests, it is important to optimise them.

6.1 The DBpedia Project

As mentioned in Chapter 1, knowledge bases play an increasingly important role in enhancing the intelligence of Web and enterprise search, as well as in supporting information integration. Today, most knowledge bases cover only specific domains, are created by relatively small groups of knowledge engineers, and are very cost intensive to keep up-to-date as knowledge changes. At the same time, Wikipedia has grown into one of the central knowledge sources of mankind, maintained by thousands of contributors.

¹The author is co-founder and actively contributing to the DBpedia project. The work presented in this section was done in co-operation with the Free University Berlin and the company OpenLink.

The DBpedia project leverages this source of knowledge by extracting structured information from Wikipedia and making this information accessible on the Web. The resulting DBpedia knowledge base currently (as of release 3.4) describes more than 2.9 million entities, including 282,000 persons, 339,000 places and 119,000 organisations. The knowledge base contains 3.8 million links to external web pages; and 4.9 million RDF links into other Web data sources. The DBpedia knowledge base has several advantages over existing knowledge bases: It covers many domains, it represents real community agreement, it automatically evolves as Wikipedia changes, it is truly multilingual, and it is accessible on the Web.

For each entity, DBpedia provides a globally unique identifier that can be dereferenced according to the Linked Data principles². As DBpedia covers a wide range of domains and has a high degree of conceptual overlap with various open-license datasets that are already available on the Web, an increasing number of data publishers have started to set RDF links from their data sources to DBpedia, making DBpedia one of the central interlinking hubs of the emerging Web of Data.

6.1.1 The DBpedia Knowledge Extraction Framework

Wikipedia articles consist mostly of free text, but also contain various types of structured information in the form of wiki markup. Such information includes infobox templates, categorisation information, images, geo-coordinates, links to external Web pages, disambiguation pages, redirects between pages, and links across different language editions of Wikipedia. The DBpedia project extracts this structured information from Wikipedia and turns it into a rich knowledge base. We give an overview of the DBpedia knowledge extraction framework, and discuss DBpedia's infobox extraction approach in more detail.

Architecture of the Extraction Framework

Figure 6.1 gives an overview of the DBpedia knowledge extraction framework. The main components of the framework are: *PageCollections* which are an abstraction of local or remote sources of Wikipedia articles, *Destinations* that store or serialize extracted RDF triples, *Extractors* which turn a specific type of wiki markup into triples, *Parsers* which support the extractors by determining datatypes, converting values between different units and splitting markup into lists. *ExtractionJobs* group a page collection, extractors and a destination into a workflow. The core of the framework is the *Extraction Manager* which manages the process of passing Wikipedia articles to the extractors and delivers their output to the destination. The Extraction Manager also handles URI management and resolves redirects between articles.

The framework currently consists of extractors which process the following types of Wikipedia content:

²<http://www.w3.org/DesignIssues/LinkedData.html>, <http://sites.wiwiiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/>

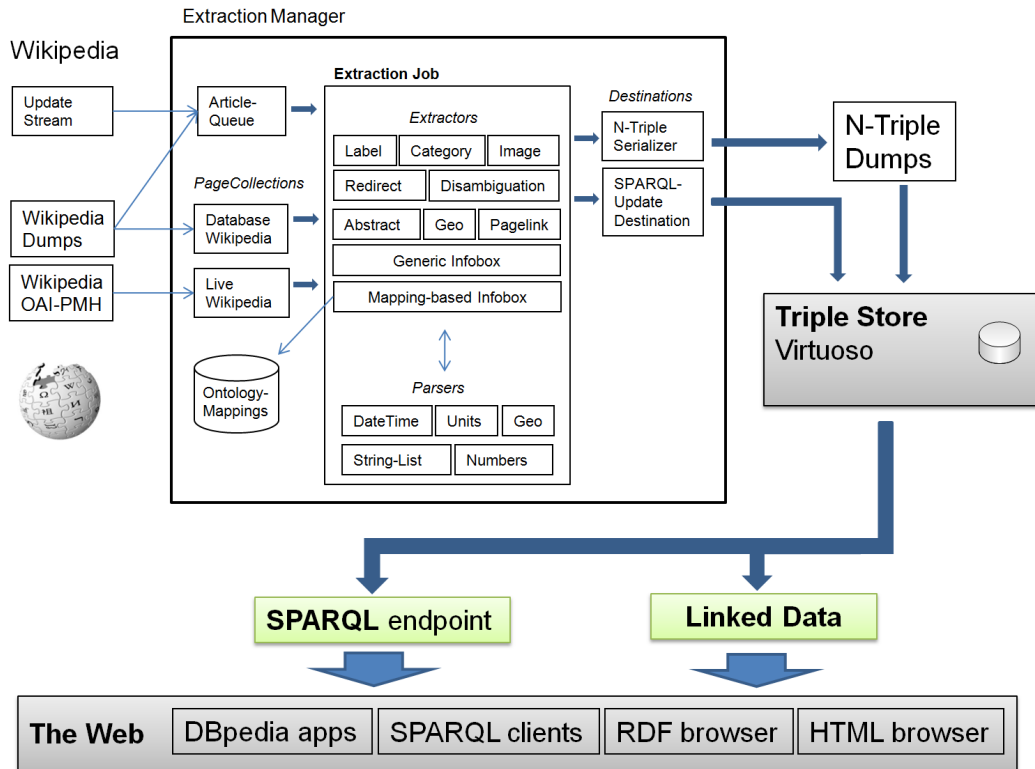


Figure 6.1: Overview of DBpedia components.

- *Infoboxes.* Wikipedia articles often have infoboxes summarising the most important information regarding the described subject. This information is the core of DBpedia and its extraction is described in more detail below.
- *Labels.* All Wikipedia articles have a title, which is used as an `rdfs:label` for the corresponding DBpedia resource.
- *Abstracts.* We extract a short abstract (first paragraph, represented using `rdfs:comment`) and a long abstract (text before a table of contents, at most 500 words, using the property `dbpedia:abstract`) from each article.
- *Interlanguage links.* We extract links that connect articles about the same topic in different language editions of Wikipedia and use them for assigning labels and abstracts in different languages to DBpedia resources.
- *Images.* Links pointing at Wikimedia Commons images depicting a resource are extracted and represented using the `foaf:depiction` property.
- *Redirects.* In order to identify synonymous terms, Wikipedia articles can redirect to other articles. We extract these redirects and use them to resolve references between DBpedia resources.

- *Disambiguation*. Wikipedia disambiguation pages explain the different meanings of homonyms. We extract and represent disambiguation links using the predicate `dbpedia:disambiguates`.
- *External links*. Articles contain references to external Web resources which we represent using the DBpedia property `dbpedia:reference`.
- *Pagelinks*. We extract all links between Wikipedia articles and represent them using the `dbpedia:wikilink` property.
- *Homepages*. This extractor obtains links to the homepages of entities such as companies and organisations by looking for the terms *homepage* or *website* within article links (represented using `foaf:homepage`).
- *Categories*. Wikipedia articles are arranged in categories, which we represent using the SKOS vocabulary³. Categories become `skos:concepts`; category relations are represented using `skos:broader`.
- *Geo-coordinates*. The geo-extractor expresses coordinates using the Basic Geo (WGS84 lat/long) Vocabulary⁴ and the GeoRSS Simple encoding of the W3C Geospatial Vocabulary⁵. The former expresses latitude and longitude components as separate facts, which allows for simple areal filtering in SPARQL queries.

The DBpedia extraction framework is currently set up to realize two workflows: A regular, dump-based extraction and a live extraction.

Dump-based extraction. The Wikimedia Foundation publishes SQL dumps of all Wikipedia editions on a monthly basis. We regularly update the DBpedia knowledge base with the dumps of 30 Wikipedia editions. The dump-based workflow uses the *DatabaseWikipedia page collection* as the source of article texts and the N-Triples serializer as the output destination. The resulting knowledge base is made available as Linked Data, for download, and via DBpedia’s main SPARQL endpoint.

Live Extraction. The Wikimedia Foundation has given the DBpedia project access to the *Wikipedia OAI-PMH live feed* that instantly reports all Wikipedia changes. The live extraction workflow uses this update stream to extract new RDF whenever a Wikipedia article is changed. The text of these articles is accessed via the *LiveWikipedia page collection*, which obtains the current article version encoded according to the OAI-PMH protocol. The *SPARQL-Update Destination* deletes existing and inserts new triples into a separate triple store. According to our measurements, about 1.4 article pages are updated per second on the English Wikipedia. The framework can handle up to 3.5 pages per second on a 2.4 GHz

³<http://www.w3.org/2004/02/skos/>

⁴<http://www.w3.org/2003/01/geo/>

⁵<http://www.w3.org/2005/Incubator/geo/XGR-geo/>

```

{{ Infobox Actor
| birthname = Thomas Jeffrey Hanks
| birthdate = {{birth date and age|1956|7|9}}
| birthplace = [[Concord, California|Concord]], [[California]]
| yearsactive = 1979 - present
| occupation = Actor, producer, director, [[voice over artist]],
               writer, speaker  }}

```

Figure 6.2: Infobox Tom Hanks.

dual-core machine (this includes consumption from the stream, extraction, computing the diff and loading the triples into a Virtuoso triple store). The time lag for DBpedia to reflect Wikipedia changes is currently about 5 minutes. The changes are not processed immediately, since empirically a second article edit follows the first edit within a few minutes. The live extraction allows wiki users to modify the schema underlying DBpedia. More information about this can be found at <http://meta.wikimedia.org/wiki/DBpedia/ontology>.

Generic versus Mapping-based Infobox Extraction

The type of wiki contents that is most valuable for the DBpedia extraction are Wikipedia infoboxes. Infoboxes display an article's most relevant facts as a table of attribute-value pairs on the top right-hand side of the Wikipedia page. Figure 6.2 shows excerpts of the wiki markup behind the infobox describing Tom Hanks. Wikipedia's infobox template system has evolved over time without central coordination. Different communities use different templates to describe closely related things (e.g. `infobox_city_japan`, `infobox_swiss_town` and `infobox_town_de`). Different templates use different names for the same attribute (e.g. `birthplace` and `placeofbirth`). As many Wikipedia editors do not strictly follow the recommendations given on the page that describes a template, attribute values are expressed using a wide range of different formats and units of measurement. The DBpedia project has decided to deal with this situation by using two different extraction approaches in parallel: A generic approach which aims at wide coverage and a mapping-based approach which aims at high data quality.

Generic Infobox Extraction. The generic infobox extraction algorithm, which is described in detail in [Auer and Lehmann, 2007], processes all infoboxes within a Wikipedia article. It creates triples from the infobox data in the following manner: The corresponding DBpedia URI of the Wikipedia article is used as subject. The predicate URI is created by concatenating the namespace fragment `http://dbpedia.org/property/` and the name of the infobox attribute. Objects are created from the attribute value. Property values are post-processed in order to generate suitable URI references or literal values. This includes recognizing MediaWiki links, detecting lists, and using units as datatypes. MediaWiki templates

may be nested, which is handled through a blanknode creation algorithm. The advantage of the generic extraction is its complete coverage of all infoboxes and infobox attributes. The main disadvantage is that synonymous attribute names are not resolved, which makes writing queries against generic infobox data rather cumbersome. As Wikipedia attributes do not have explicitly defined datatypes, a further problem is the relatively high error rate of the heuristics that are used to determine the datatypes of attribute values.

Mapping-based Infobox Extraction. In order to overcome the problems of synonymous attribute names and multiple templates being used for the same type of things, we mapped Wikipedia templates to an ontology. This ontology was created by manually arranging the 350 most commonly used infobox templates within the English edition of Wikipedia into a subsumption hierarchy consisting of 170 classes and then mapping 2350 attributes from within these templates to 720 ontology properties. The property mappings define fine-grained rules on how to parse infobox values and define target datatypes, which help the parsers to process attribute values. For instance, if a mapping defines the target datatype to be a list of links, the parser will ignore additional text that might be present in the attribute value. The ontology currently uses 55 different datatypes. Deviant units of measurement are normalized to one of these datatypes. Instance data within the infobox ontology is therefore cleaner and better structured than data that is generated using the generic extraction algorithm. The disadvantage of the mapping-based approach is that it currently covers only 350 Wikipedia templates; therefore it only provides data about 843,000 entities compared to 1,462,000 entities that are covered by the generic approach. While the ontology is currently relatively simple, we plan to extend it further, e.g. with class disjointness axioms and inverse properties. The main purpose of such extensions will be to allow consistency checks in DBpedia and use inferences when answering SPARQL queries. As noted previously, we are currently working on crowd-sourcing the mapping creation task in the context of the DBpedia live extraction.

6.1.2 The DBpedia Knowledge Base

Table 6.1 gives an overview of common DBpedia classes, and shows the number of instances and some example properties for each class. In the following, we describe the structure of the DBpedia knowledge base, explain how identifiers are built and compare the four classification schemata that are offered by DBpedia.

Identifying Entities

DBpedia uses English article names for creating identifiers. Information from other language versions of Wikipedia is mapped to these identifiers by bidirectionally evaluating the interlanguage links between Wikipedia articles. Resources are assigned a URI according to the pattern `http://dbpedia.org/resource/Name`, where *Name* is taken from the URL of the source Wikipedia article, which has

Ontology Class	Instances	Example Properties
Person	198,056	name, birthdate, birthplace, employer, spouse
Artist	54,262	activeyears, awards, occupation, genre
Actor	26,009	academyaward, goldenglobeaward, activeyears
MusicalArtist	19,535	genre, instrument, label, voiceType
Athlete	74,832	currentTeam, currentPosition, currentNumber
Politician	12,874	predecessor, successor, party
Place	247,507	lat, long
Building	23,304	architect, location, openingdate, style
Airport	7,971	location, owner, IATA, lat, long
Bridge	1,420	crosses, mainspan, openingdate, length
Skyscraper	2,028	developer, engineer, height, architect, cost
PopulatedPlace	181,847	foundingdate, language, area, population
River	10,797	sourceMountain, length, mouth, maxDepth
Organisation	91,275	location, foundationdate, keyperson
Band	14,952	currentMembers, foundation, homeTown, label
Company	20,173	industry, products, netincome, revenue
Educ.Institution	21,052	dean, director, graduates, staff, students
Work	189,620	author, genre, language
Book	15,677	isbn, publisher, pages, author, mediatype
Film	34,680	director, producer, starring, budget, released
MusicalWork	101,985	runtime, artist, label, producer
Album	74,055	artist, label, genre, runtime, producer, cover
Single	24,597	album, format, releaseDate, band, runtime
Software	5,652	developer, language, platform, license
TelevisionShow	10,169	network, producer, episodenummer, theme

Table 6.1: Common DBpedia classes with the number of their instances and example properties.

the form `http://en.wikipedia.org/wiki/Name`. This yields certain beneficial properties:

- DBpedia URIs cover a wide range of encyclopaedic topics.
- They are defined by community consensus.
- There are clear policies in place for their management.
- An extensive textual definition of the entity is available at a well-known Web location (the Wikipedia page).

Classifying Entities

DBpedia entities are classified within four classification schemata in order to fulfil different application requirements. We compare these schemata below:

Wikipedia Categories. DBpedia contains a SKOS representation of the Wikipedia category system. The category system consists of 415,000 categories. The main advantage of the category system is that it is collaboratively extended and kept up-to-date by thousands of Wikipedia editors. A disadvantage is that categories do not form a proper topical hierarchy, as there are cycles in the category system and as categories often only represent a rather loose relatedness between articles.

YAGO. The YAGO classification schema consists of 286,000 classes which form a deep subsumption hierarchy. The schema was created by mapping Wikipedia leaf categories, i.e. those not having subcategories, to WordNet synsets. Details of the mapping algorithm are described in [Suchanek et al., 2008]. Characteristics of the YAGO hierarchy are its deepness and the encoding of much information in one class (e.g. the class “MultinationalCompaniesHeadquarteredInTheNetherlands”). While YAGO achieves a high accuracy in general, there are a few errors and omissions (e.g. the mentioned class is not a subclass of “MultinationalCompanies”) due to its automatic generation. We jointly developed a script that assigns YAGO classes to DBpedia entities. The script is available at the YAGO download page⁶.

UMBEL. The Upper Mapping and Binding Exchange Layer (UMBEL) is an ontology that has been created for interlinking Web content and data. UMBEL was derived from *OpenCyc* and consists of 20,000 classes. OpenCyc classes in turn are partially derived from Cyc collections, which are based on WordNet synsets. Since YAGO also uses WordNet synsets and is based on Wikipedia, a mapping from OpenCyc classes to DBpedia can be derived via UMBEL⁷.

⁶<http://www.mpi-inf.mpg.de/yago-naga/yago/downloads.html>

⁷<http://fgiasson.com/blog/index.php/2008/09/04/exploding-dbpedias-domain-using-umbel/>

The classification is maintained by the UMBEL project itself and details about its generation process can be found at the UMBEL website⁸.

DBpedia Ontology. The DBpedia ontology consists of 170 classes that form a shallow subsumption hierarchy. It includes 720 properties with domain and range definitions. The ontology was manually created from the most commonly used infobox templates within the English edition of Wikipedia. The ontology is used as target schema by the mapping-based infobox extraction described in Section 6.1.1. The left column in Table 6.1 displays a part of the class hierarchy of the DBpedia ontology.

Graph Characteristics

	Described Entities	Mio. Triples	Unique Properties	Triples/ Property	Triples/ Entity
Generic					
Extract.	1,462,108	26.0	38,659	673.7	17.81
Mapping- based	843,169	7.0	720	9722.2	8.34
Pagelinks	2,853,315	70.2	1	70.2mio	24.61

Table 6.2: Comparison of the generic infobox, mapping-based infobox and pagelinks datasets.

	Connected Entities	Mio. Triples	Unique Properties	Indegree Max	Avg	Cluster Coefficient
Generic						
Extract.	1,029,712	5.6	9911	105,840	8.76	0.1336
Mapping- based	627,941	2.9	340	65,387	11.03	0.1037
Pagelinks	2,796,401	46.2	1	190,995	19.15	0.1696

Table 6.3: Comparison of the graph structure of the generic infobox, mapping-based infobox and pagelinks datasets.

Table 6.2 compares the datasets that result from generic infobox extraction, the mapping-based infobox extraction and from the extraction of links between Wikipedia pages (statistics are for the DBpedia release 3.2, English data sets). It is evident that properties are, of course, reused very often in the mapping-based approach. We then measured further characteristics of the DBpedia RDF graph. For doing this, we removed all triples from the datasets that did not point at a

⁸<http://www.umbel.org/>

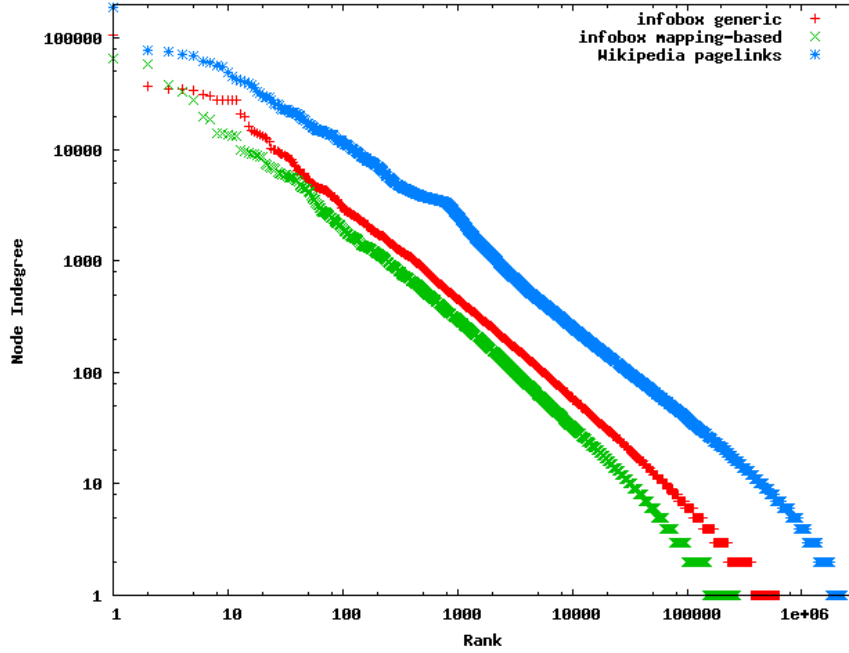


Figure 6.3: Comparison of the generic infobox, mapping-based infobox and pagelinks datasets in terms of node indegree versus rank.

DBpedia entity, including all literal triples, all external links and all dead links. The size of and number of 'link' properties within these reduced datasets is listed in Table 6.3. Removing the triples showed, that the percentage of properties pointing to other DBpedia entities is much higher in the mapping-based dataset (53%) compared to generic dataset (25.6%). We calculated the average node indegree as the sum of all inbound edges divided by the number of objects, which had at least one inbound edge from the dataset. This allows to analyse the indegree separately from the coverage or the size of the dataset. The entity with the highest indegree within all three datasets is *United States*. As shown in Figure 6.3, the node indegrees follow a power-law distribution in all datasets which is a typical characteristic of small world networks [Reka and Albert-Laszlo, 2002]. The clustering coefficient given in the last column of Table 6.3 was calculated as the number of existing connections between neighbors of a node, divided by possible connections in a directed graph ($k * (k - 1)$, k = number of node neighbors) and averaged over all nodes. The mapping-based approach has a slightly lower clustering coefficient because of its lower coverage.

6.1.3 Interlinked Web Content

The DBpedia knowledge base is interlinked with various other data sources on the Web according to the Linked Data principles. These links provide the basis for browsing [T. Berners-Lee et al., 2006, Becker and Bizer, 2008], crawling and


```

<http://dbpedia.org/resource/Spain> owl:sameAs
  http://rdf.freebase.com/ns/guid.9202a8c04000641f8000000000034e30;
  http://[...]fu-berlin.de/factbook/resource/Spain;
  http://[...]fu-berlin.de/eurostat/resource/countries/Espa%C3%B1a;
  http://sw.opencyc.org/2008/06/10/concept/Mx4rvVjowpwpEbGdrcN5Y29ycA.

<http://data.semanticweb.org/conference/eswc/2008/paper/356>
  swc:hasTopic <http://dbpedia.org/resource/Data_integration> .

```

Figure 6.4: Example RDF links connecting the DBpedia entity *Spain* with additional information from other data sources, and showing how the DBpedia identifier *Data Integration* is used to annotate the topic of a conference paper.

searching [Cheng et al., 2008, Harth et al., 2008, Tummarello et al., 2007], building mashups [Naumann et al., 2006], usage for content annotation etc. Figure 6.4 shows RDF data links that illustrate these use cases. The first four links connect the DBpedia entity *Spain* with complementary data about the country from EuroStat, the CIA World Factbook, Freebase and OpenCyc. Agents can follow these links to retrieve additional information about Spain, which again might contain further deeper links into the data sources. The fifth link illustrates how the DBpedia identifier *Data Integration* is used to annotate the topical subject of a research paper from the European Semantic Web Conference. After this and similar annotations from other sites have been crawled by a search engine, such links enable the discovery of Web content that is related to a topic.

Figure 6.5 gives an overview of the data sources that are currently interlinked with DBpedia. Altogether this Web of Data amounts to approximately 4.7 billion RDF triples. Two billion of these triples are served by data sources participating in the *W3C Linking Open Data community project*⁹, an effort to make open-license datasets interoperable on the Web of Data by converting them into RDF and by interlinking them. The success of the Linking Open Data initiative and the size of the incorporated knowledge bases motivate the fragment extraction approach in Section 6.2, which enables the use of learning algorithms on those knowledge bases.

Table 6.4 lists the data sources that are reachable from DBpedia by outgoing RDF links¹⁰. The second column shows the distribution of the 4.9 million outgoing links over the data sources. Using these links, one can, for instance, navigate from a computer scientist in DBpedia to her publications in the DBLP database, from

⁹<http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData>

¹⁰For more information about the datasets please refer to <http://wiki.dbpedia.org/Interlinking>

Data Source	Classes	Data Source	Classes
BBC Music	musicians, bands	LIBRIS	authors
Bio2RDF	genes, proteins, molecules	LinkedCT	intervention, conditions
CrunchBase	companies	Linked Drug-Bank	drugs, diseases
Diseasome	diseases	LinkedMDB	films
Faviki	various classes	Lingvoj	languages
flickr wrapppr	various classes	OpenCyc	various classes
FOAF	various classes	OpenCalais	locations, people
GeoNames	places	Surge Radio	musicians, bands
GeoSpecies	species	UMBEL	various classes
John Peel	musicians, works	RDFohloh	programming languages
		Revyu	various classes
		LODD SIDER	drug side effects
		Semantic Web Corpus	various classes

Table 6.5: Data sources publishing RDF links pointing at DBpedia entities.

that have been crawled from the Web by the Sindice Semantic Web Search engine [Tummarello et al., 2007]. The analysis revealed that there are currently 23 external data sources setting RDF links to DBpedia. Table 6.5 lists these data sources together with the classes of DBpedia entities that are the targets of the incoming links.

6.1.4 Applications

The DBpedia knowledge base and the Web of Data around DBpedia lay the foundation for a broad range of applications. As DBpedia is interlinked with various other data sources, DBpedia URIs are good starting points to explore or crawl the Web of Data. Data browsers that can be used to explore the Web of Data include Tabulator [T. Berners-Lee et al., 2006], Marbles¹¹, Disco¹², and the Open-Link Data Explorer¹³.

DBpedia Mobile [Becker and Bizer, 2008] is a location-aware client for the Semantic Web that uses DBpedia locations as navigation starting points. DBpedia

¹¹<http://becker.org/marbles>

¹²<http://sites.wiwiss.fu-berlin.de/suhl/bizer/ng4j/disco/>

¹³<http://ode.openlinksw.com/example.html>

Mobile¹⁴ allows users to discover, search and publish Linked Data pertaining to their current physical environment using an iPhone and other mobile devices as well as standard web browsers. Based on the current GPS position of a mobile device, DBpedia Mobile renders an interactive map indicating nearby locations from the DBpedia dataset. DBpedia Query Builder¹⁵ is a tool to demonstrate the querying capabilities of DBpedia. It allows expert and non-expert users to formulate and execute queries over DBpedia and save them for later use.

Muddy Boots¹⁶ is a project commissioned by the BBC that aims to enhance the BBC news stories with external data. The Muddyboots APIs allow to identify the main actors (people and companies) in a BBC news story in an unambiguous way by means of DBpedia identifiers. In this way, the story is linked to DBpedia data, which is then used by a BBC prototype to populate a sidebar with background information on identified actors [Kobilarov et al., 2009]. Open Calais¹⁷ is a project by Thomson Reuters that provides a web service for named entity recognition from freetext as well as related tools. With release 4 of the web service, entity descriptors are published as Linked Data with outgoing `owl:sameAs` links to DBpedia, Freebase and GeoNames. Faviki¹⁸ is a social bookmarking tool that allows tagging of bookmarks with Wikipedia-based identifiers to prevent ambiguities. Identifiers are automatically suggested using the Zemanta API (see below). DBpedia is leveraged to view tags by topics and to provide tag descriptions in different languages. Zemanta¹⁹ provides tools for the semi-automated enrichment of blogs. The company offers its annotation engine to third parties via an API. Zemanta recently extended its API to generate RDF links pointing at DBpedia, Freebase, MusicBrainz and Semantic CrunchBase. LODr²⁰ allows users to tag content that they contributed to popular Web 2.0 services (Flickr, del.icio.us, slideshare) using Linked Data identifiers, such as those provided by DBpedia. Topbraid Composer²¹ is a Semantic Web modeling environment that includes a built-in capability to resolve a label to a Wikipedia article, from which it derives a DBpedia resource URI.

Two applications developed by the author in corporation with other researchers (see [Lehmann et al., 2007, Heim et al., 2009, Lehmann and Knappe, 2008]) are the DBpedia Relationship Finder and the DBpedia Navigator, which are briefly described in the sequel.

¹⁴<http://beckr.org/DBpediaMobile>

¹⁵<http://querybuilder.dbpedia.org/>

¹⁶<http://muddyboots.rattlerresearch.com/>

¹⁷<http://opencalais.com/>

¹⁸<http://faviki.com>

¹⁹<http://zemanta.com>

²⁰<http://lodr.info/>

²¹<http://www.topbraidcomposer.com/>

RelFinder A user interface that can be used to explore the DBpedia knowledge base is the DBpedia Relationship Finder²². The Relationship Finder allows users to find connections between two different entities in DBpedia. The Relationship Finder user interface initially contains a simple form to enter two entities, as well as a small number of options, and a list of previously saved queries. While typing, the user is offered suggestions for the object he wants to enter. The first version of the DBpedia Relationship Finder was published in [Lehmann et al., 2007] along with statistical discoveries in DBpedia.

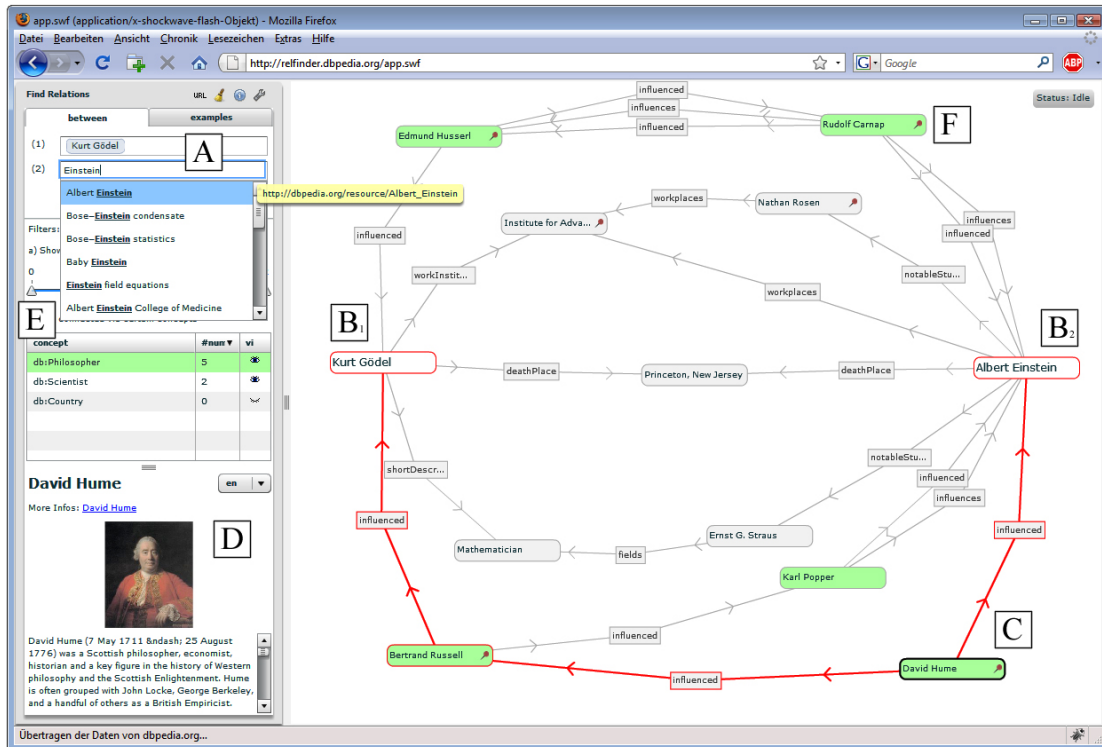


Figure 6.6: Revealing relationships between Kurt Gödel und Albert Einstein.

In [Heim et al., 2009] the tool was renamed to *RelFinder* and generalised to arbitrary knowledge bases accessible via SPARQL, while DBpedia remained its most popular use case. Figure 6.6 gives a quick overview of its functionality. The search terms that are entered by the user in the two input fields in the upper left corner (Fig. 6.6, A) get mapped to unique objects of the knowledge base. These constitute the left and right starting nodes in the graph visualization (Fig. 6.6, B) that get then connected by relations and objects found in between them by the algorithm. If a certain node is selected, all graph elements that connect this node with the starting nodes are highlighted forming one or more paths through the graph (Fig. 6.6, C). In addition, further information about the selected object is displayed in the sidebar (Fig. 6.6, D). Filters can be applied to increase or reduce

²²<http://relfinder.dbpedia.org/>

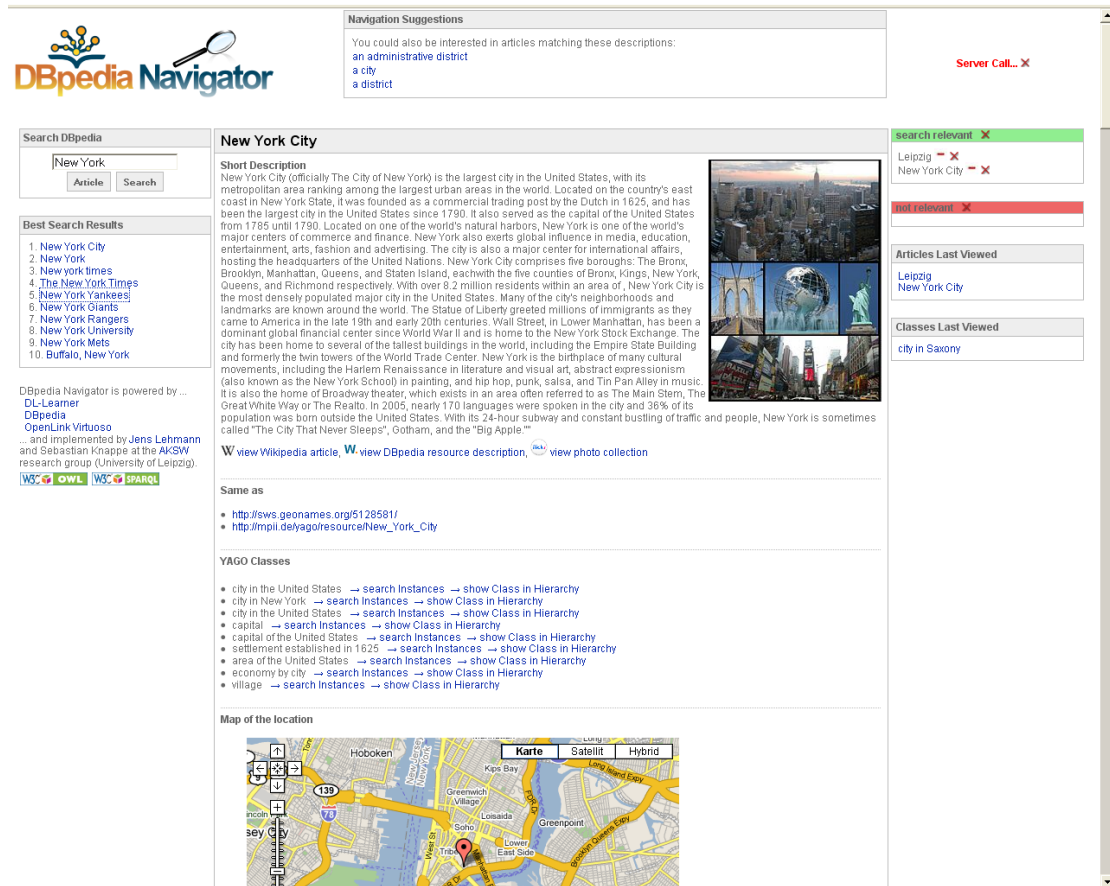


Figure 6.7: Overview of the DBpedia Navigator GUI. DBpedia Navigator provides an interface for searching and browsing within DBpedia.

the number of relationships that are shown in the graph and to focus on certain aspects of interest (Fig. 6.6, E). We refer the interested reader to the article for a description of how the RelFinder is realised technically.

DBpedia Navigator DBpedia Navigator²³ is a user interface for browsing, searching, and navigating within DBpedia. For searching entities, it uses the Virtuoso triple store full text index. If there is no match, a list of resources containing the string will be shown. The list is ordered by the number of DBpedia pagelinks to an article, which is a simple but effective way to rate search results. The number of pagelinks has been computed beforehand, such that search results can be ranked very fast.

The entity view is tailored for DBpedia and consists of several parts, which are separated by a dotted horizontal line. The first part shows a short description of the object with an associated picture. Links to the corresponding Wikipedia article, the DBpedia resource description and a Flickr photo collection are given.

²³<http://navigator.dbpedia.org>

The second part consists of links to interesting interlinked objects from other knowledge bases. Below those, the classes of the object are depicted. The used class hierarchy can be switched between YAGO and the DBpedia ontology in a configuration file. Using the displayed links, you can search for instances of these classes, which takes you to a search result view of the instances, or alternatively you can view information about the class itself. The third part is a content-specific section depending on the kind of viewed object. If the object is a location, a Google Map is shown, if it is a person, some characteristics of that person are displayed etc. Finally, the fourth part is a collection of interesting information, which was not yet consumed by any of the parts above (interlinked data, class hierarchy, short abstract etc.) and is not ignored by a configurable object filter (e.g. we filtered out SKOS properties). They are displayed in a style similar to typical Linked Data browsers. Properties with many values are grouped.

Shown articles are automatically added to the list of search relevant articles in the upper box on the right sidebar. These instances are used to generate navigation suggestions (see top of screenshot) via the presented OCEL machine learning algorithm in combination with the fragment extraction approach presented in the next section.

In [Auer and Lehmann, 2007, Auer et al., 2008, Lehmann et al., 2009] the interested reader can study more technical details about DBpedia and a discussion of related work.

6.2 Knowledge Fragment Selection

We currently experience that Semantic Web technologies are gaining momentum and large knowledge bases such as the described DBpedia, OpenCyc [Lenat, 1995], GovTrack²⁴ and others are freely available (see Figure 6.5). These knowledge bases are based on semantic web knowledge representation standards. They contain hundred thousands of properties as well as classes and an even larger number of facts and relationships. These knowledge bases and many more²⁵ are available as Linked Data or SPARQL endpoints [Clark et al., 2008].

The learning algorithms we introduced are appropriate for small and medium size knowledge bases, while they cannot be directly applied to large knowledge bases (such as the initially mentioned ones) due to their dependency on reasoning methods. We present an approach for leveraging those algorithms. The scalability of the algorithms is ensured by reasoning only over "interesting parts" of a knowledge base for a given task.

We present the following results (for details see [Hellmann et al., 2009a] as our main reference for this section):

²⁴<http://www.govtrack.us>

²⁵<http://esw.w3.org/topic/SpqrqlEndpoints>

- development of a flexible method for extracting relevant parts of very large and possibly interlinked knowledge bases for a given learning task,
- thorough implementation, integration, and evaluation of these methods in the DL-Learner framework (see Section 7.1)
- presentation of several application scenarios and examples employing some very large knowledge bases available on the Web.

Motivation The most commonly used reasoners such as Pellet and Fact++ do not, although highly optimized and efficient, have the ability to scale up to large knowledge bases. Thus, it becomes impossible to use the presented learning algorithms as soon as the target knowledge base reaches a certain size and complexity, with two major problems being initialization time (i.e. to load the data into the reasoner) and the time to answer instance checks. However, in order to solve the learning problem it is often not necessary to consider the complete knowledge base, but only a fragment that holds enough information to produce good results, while at the same time is small enough to allow efficient reasoning.

Desired Fragment We are looking for a sufficiently small fragment F of an ontology O ($F \subseteq O$), which contains the examples E and further relevant information to solve a given learning problem LP . If we can successfully apply the learning algorithm on the fragment yielding the concept C , which is a solution of the learning problem, then C should also be a solution of the learning problem in the large knowledge base O .

The following example shall briefly illustrate, what can be achieved by our fragment selection approach, before we will explain, in detail, how such a fragment is selected and which parameters are used.

Example 6.1 (Manual Example From Semantic Bible)

Here and also in later experiments, we choose the Semantic Bible ontology²⁶, because it is a medium sized ontology, contains rich background knowledge and is still manageable by a reasoner as a whole. This enables us to directly compare the results of learning on the fragment to results obtained on the whole knowledge base. We manually choose Archelaus and HerodAntipas, two brothers from the New Testament as positive examples, while we choose God, Jesus, Michael and Gabriel (the archangels) as negative examples. The learning algorithm was then executed twice, once in normal mode, where the whole ontology was loaded into the OWL reasoner (Pellet) and once where first a fragment was selected by our extraction method²⁷, which was then loaded into Pellet (see Figure 6.8 for an overview). The 20 best learned expressions from both runs (like $\exists \text{ siblingOf.Man}$ or $\exists \text{ siblingOf}.\exists \text{ spouseOf.Human}$) are with some exceptions identical and, even

²⁶<http://www.semanticbible.com/ntn/ntn-overview.html>

²⁷The ontology was loaded into a local Joseki triple store and queried with SPARQL.

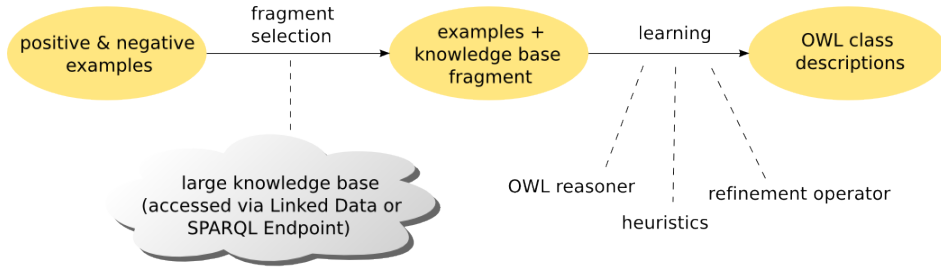


Figure 6.8: Process illustration: In a first step, a fragment is selected based on objects from a knowledge source and in a second step the learning process is started on this fragment and the given examples.

Semantic Bible	Normal	Fragment
No. of classes	49	27
No. of objects	724	60
No. of object properties	29	20
No. of data properties	9	0
No. of subclass axioms	51	25
Time needed for extraction	-	4.2s
Reasoner instantiation time	3.6s	1.3s
No. of reasoner queries	1480	313
Avg. time per query	120ms	2ms
Reasoning time	178.0s	0.8s
Learning time without reasoning	0.4s	0.1s
Total time	182.0s	6.4s

Table 6.6: Manual example to give a first glance at the presented method. Note that not only are reasoner queries faster on average, but also the number of queries needed is significantly smaller (due to the smaller search space.)

more important, all 20 classes learned from the fragment yield 100% accuracy on the whole ontology. Table 6.6 provides details on the Semantic Bible ontology and solving the learning problem on it as a whole or on a fragment.

6.2.1 What Properties Should the Fragment Have?

In the previous section, we stated what a desired fragment is. It allows fast reasoning and the learned expressions achieve (approximately) the same accuracy when validated versus the original knowledge base. We now take a closer look at what should be included in the fragment for the learning algorithm to work efficiently while still achieving good results. The first obvious inclusions are the

example objects themselves. Secondly, all classes of the examples and all related objects (via an object property) are necessary. Note that the property between objects will always be included implicitly, when we add related objects to the fragment. Up to now, the fragment consists of the combined Concise Bound Descriptions (CBD²⁸) of the examples. The information contained is clearly not yet sufficient to learn complex classes. Some class expressions derivable when using only CBDs are of the form $C \sqcup R$ or $C \sqcap R$, where C is any conjunction or disjunction of named classes and R is a conjunction or intersection of unqualified property restrictions of the form $\exists \text{property}.\top$. While this is of course often not sufficient, it still represents the smallest sensible fragment, where it is possible to learn expressions at all with the trade-off scale shifted away from high learning accuracy towards efficient light-weight reasoning.

One of the major influences on the validity of learning results stands in direct relation to the possible deductive inferences on the fragment. Since reasoning in description logics is monotonic, the inferences obtained on a fragment of an ontology are also valid for the ontology as a whole (soundness). However, not all possible inferences might be obtainable on the fragment and reasoning thus can be viewed as being “incomplete”, e.g. an instance check $C(a)$ answered negatively on the fragment (the reasoner cannot deduce that a is instance of C) might be answered positively on the whole ontology.

As a consequence, a learning problem might be solved incorrectly, because the learning algorithm implicitly assumes that the underlying reasoning methods are complete. So, if for a class expression C the resulting answer set of a retrieval will contain all positive examples and none of the negatives, it will present C as a solution. Due to the issues explained above, however, the previously not covered negative example individuals might now be an instance of C when the whole ontology is considered. Thus a correctly learned class expression might turn out to be inconsistent. We tackle this problem by trying to avoid such cases through selection of an ontology fragment containing all relevant information as described in detail below. Furthermore, in most application scenarios the learned class expressions (and/or its implications) are reviewed by a human expert. Since reasoning on very large knowledge bases is currently almost impossible, it is hard to give exact measures of the extend to which the negative example coverage problem occurs in those cases. However, we will later perform further benchmarks on the medium sized Semantic Bible ontology and can draw conclusions from those observations.

6.2.2 Extending Concise Bound Descriptions (CBDs)

In the following, we will give a list containing which information can be additionally extracted to learn more complex classes compared to CBDs. We assume that the CBDs of all example individuals are already included in the fragment. On

²⁸<http://www.w3.org/Submission/CBD/>

this basis, the following list shows additional information which can be extracted:

1. *Direct Classes* Retrieving direct classes for all objects a in the fragment, that do not yet have any types, i.e. an assertion of the form $C(a)$, will allow to learn qualified property restrictions of the form $\exists \text{property.C}$.
2. *Increased Property Depth* A further extension of the CDBs by objects, which are related to an object, which is related to an example etc., enables to learn expressions with nested property restrictions, e.g. $\exists \text{propA}.\exists \text{propB}.\top$. This extension can be continued such that it is possible to learn nested property restrictions up to some recursion depth as parameter of the extraction algorithm.
3. *Hierarchy* Retrieving the class hierarchy relevant for the fragment improves the efficiency of the learning algorithm, because it 1) optimizes the search tree with the help of the subsumption hierarchy and 2) enables the usage of those classes in learned descriptions. The algorithm performs this step by querying for all super classes of existing classes in the fragment until no new classes are obtained.
4. *Class Definitions and Axioms* Extracting information for all classes in the fragment, e.g. definitions (`owl:equivalentClass`), disjointness, etc., will permit the learning algorithm to make use of this valuable background knowledge. For instance, knowing whether classes are disjoint speeds up the reasoning and learning process. Definitions are, of course, necessary to draw inferences relevant for instance checks. In general, extracting class related axioms reduces the above mentioned negative example coverage problem.
5. *Explicit Property Information* Retrieving characteristics of object properties, such as `owl:SymmetricProperty`, domain/range, and the property hierarchy allows useful inferences by the OWL reasoner.
6. *Inferred Property Information* Because reasoning is normally deactivated in SPARQL endpoints or Linked Data sources, reasoning on the fragment could be improved by also including objects that are related to the examples via a symmetric, inverse or transitive property. Nevertheless to include such properties, which only become “visible” after inference, additional costly queries need to be used. Empirically, the negative impact on performance is considerable, so we accepted this trade-off in favor of a faster extraction procedure.
7. *Complete Class Definitions* There also is the possibility that classes which are contained in the fragment might occur somewhere in the ontology on the right hand side of a class definition (e.g. $\text{SomeClass} = \text{AnotherClass} \sqcup \text{ClassInFragment}$). As in the item above, the cost to find such information can become quite significant. To completely extract all such information

all class axioms would need to be evaluated. As above, such information requires an intensive search, which is why we refrained from including it, although we might make it available as a parameter in the future.

Another requirement for the fragment is that it should be correct OWL-DL, so that it can be processed by OWL Reasoners.

6.2.3 Extraction Methods

Although the extraction algorithm, we are about to present, was developed to fit the needs of the class learning algorithm, it can basically be applied in any context, where a set of individuals needs to be analyzed with respect to given background knowledge (a circumstance often required in Machine Learning). The size of the fragment can be controlled in a flexible way to regulate the trade-off between complete reasoning and performance. Especially the Linked Data paradigm gives rise to questions concerning reasoning and performance, which cannot merely be answered by optimizing existing reasoning algorithms and using more powerful hardware. Linked Data connects facts across knowledge bases. Due to limited computational resources, we have to decide how far links into other knowledge bases or within the knowledge base itself should be followed and how we retrieve relevant data. In the course of this section, we will describe the extraction algorithm independently from the actual knowledge source, because it is not bound to a certain data publication formalism and works for several variations such as Linked Data or SPARQL endpoints. The actual data provisioning is merely a technical implementation question. After this section though, we will describe our implementation for SPARQL endpoints, which contains optimizations of the method.

The algorithm traverses the RDF graph of the original knowledge base recursively starting from the examples. The parameters of the algorithm allow to control the size of the fragment, such that each point in the above mentioned list (information necessary to learn more complex classes) can be included or excluded. Additionally, filters are used to gain more flexibility during the extraction of the fragment. The filters are applied to the lowest possible level in the data acquisition and thus we will start with describing the acquisition interface.

Definition 6.2 (Tuple Acquisition Interface)

A tuple acquisition function $acquire_{KB}$ in the context of the described fragment extraction procedure takes four arguments: **resource** (a URI), **predicateFilter** (a list of strings), **objectFilter** (a list of strings), and **literals** (a boolean flag). The function returns tuples consisting of predicate and object of all RDF triples in \mathcal{K} where **resource** is the subject of the triple, the predicate of the triple does not start with a string contained in **predicateFilter**, the object of the triple does not start with a string contained in **objectFilter**, and the object of the triple is not a literal if **literals** is false. We will simply write $acquire(resource)$ when the context is clear. \square

The filters provide the possibility to create a fine-grained selection of the extracted information. They are especially useful for multi-domain knowledge bases such as DBpedia, where retrieving information unfiltered will lead to an unnecessary large fragment. In our case, we avoid retrieving information, that is not important to the learning process. In some cases, we do not want to use datatype properties, so they can be omitted by the literal parameter shown above. The predicate filter removes properties that are not important (e.g. when working with DBpedia we can use this to filter properties pointing to web pages and pictures). The same is true for the object filter, i.e. it filters uninteresting objects in triples.

The configuration of filter criteria is in most cases optional and is clearly content-driven. While the parameters of the extraction algorithm steer the structural selection of knowledge, filters work at a lower abstraction level. The configuration depends on the particularities of the knowledge source and the intended task and can be optimized for the application. The choice can, on the one hand, add another edge to performance and, on the other hand, allow a content-aware filtering. If the knowledge base makes use of different structural hierarchies such as DBpedia, which uses YAGO classes [Suchanek et al., 2007] and also the SKOS vocabulary²⁹ combined with its own categories, one of the hierarchies can be selected by excluding the other. Adding the SKOS namespace (<http://www.w3.org/2004/02/skos/core>) to the predicate and object filter list will guarantee that the fragment will be free of SKOS vocabulary. A Social Semantic Web application, for example, might be especially interested in FOAF and thus would filter other information.

After having defined the filters for the respective knowledge source, a recursive algorithm (see Algorithm 6) extracts relevant knowledge for each of the objects in the example set using *acquire(instance)*. The objects of the retrieved tuples (p,o) are evaluated, manipulated and used to further extract knowledge (using *acquire(o)*) until a given recursion depth is reached. The process is illustrated in Figure 6.9.

The algorithm remembers valuable information that is used to convert the fragment to OWL DL, which we will describe later.

Parameters In the following, we will relate the influences of the algorithm parameters to the list in the previous section.

The parameter *recursion depth* has the greatest influence on the number of triples extracted and included in the fragment. If set to 0 the fragment will only consist of the examples. A recursion depth of 1 means that only the directly related objects and classes are extracted, which results in the combined CBDs of all examples. A recursion depth of 2 extracts all direct classes of the examples, their direct super classes and all directly related objects and their direct classes and directly related objects. This will enable the algorithm to learn nested property restrictions (2. *Increased Property Depth*), includes some hierarchy information

²⁹<http://www.w3.org/TR/2005/WD-swbp-skos-core-spec-20051102/>

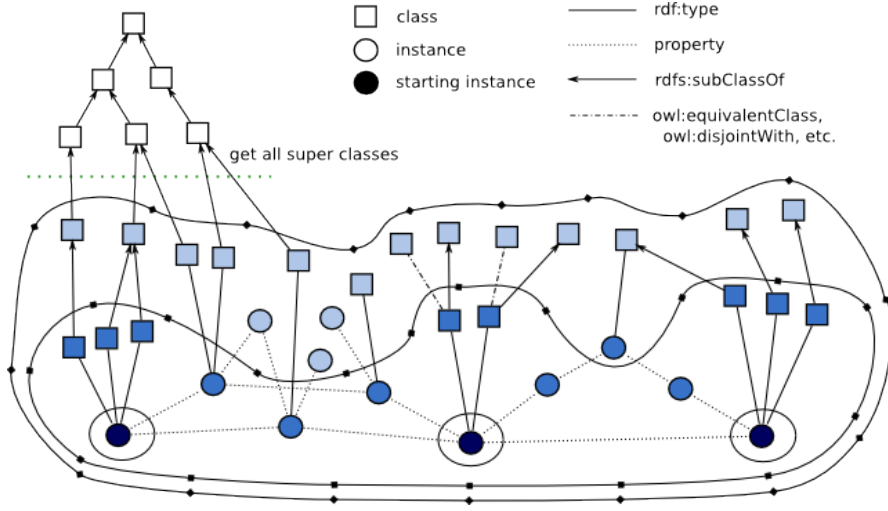


Figure 6.9: Extraction starting with three examples. The circles represent different recursion depths. The circles around the starting objects signify recursion depth 0. The larger inner circle represents the fragment with recursion depth 1 and the largest outer circle with recursion depth 2.

(3. *Hierarchy*), allows qualified property restrictions for unnested properties (1. *Direct Classes*) and includes definitions of classes directly connected to the starting individuals (4. *Class Definitions and Axioms*).

We avoid following cycles, which often occur when encountering inverse properties, `owl:sameAs`, `owl:equivalentClass` etc., by storing all resources already visited. If the object is a blank node, we will not decrease the recursion counter until no further blank nodes are retrieved. This ensures that extracted axioms, e.g. $A \equiv A_1 \sqcap A_2$, are extracted completely, which might otherwise not be the case, since their RDF serialisation consists of several triples.

If we use all existing objects of the original knowledge base as starting seeds with a sufficient recursion depth, the algorithm will extract the whole knowledge base with the exception of unconnected resources, which in most cases barely contain useful information.

To cover other points on the list above, the algorithm retrieves additional information in post-processing steps, which can be switched on and off independently.

Close after recursion For each object in the fragment that does not yet have any classes assigned to it, classes are retrieved and added to the fragment (cf list 1. *Direct Classes*).

Get all super classes For all classes in the fragment, all super classes are retrieved and the hierarchy is extracted (cf. list 3. *Hierarchy*). Additionally, all definitions are included (cf. list 4. *Class Definitions and Axioms*).

Get all property information For all object properties, types, domain, range and the property hierarchy will be retrieved (cf. list 5. *Explicit Property Information*).

Depending on the expected complexity of class expressions (in particular their

property depth) and the density of the background knowledge, a recursion depth of 1 or 2 (with all post-processing steps enabled, otherwise 2 or 3) represents a good balance between the amount of useful information and the possibility to reason efficiently in a large knowledge base.

The retrieved triples can be further manipulated by means of user defined rules. For example, vocabularies that resemble class hierarchies but use different identifiers (such as SKOS) can be mapped to OWL class hierarchies. We also used this technique to embed tags or other structurally important individuals in a class hierarchy in order to enable learning class expressions. Additional information can be easily inserted in this step of the extraction. The function `manipulate` does not only allow for manipulation, but can also be used to retrieve and include information from other knowledge bases. Even a new extraction can be started based on the current resource.

Algorithm 6: Knowledge Extraction Algorithm

```

1 Function: extract
   Input: recursion counter, resource, predicateFilter, objectFilter, literals
   Output: set  $S$  of triples
2
3 if recursion counter equals 0 then
4   return  $\emptyset$ 
5  $S$  = empty set of triples;
6 // for acquire see Definition 6.2
7 resultSet = acquire(resource, predicateFilter, objectFilter, literals);
8 newResultSet =  $\emptyset$  ;
9 foreach tuple (p,o) from resultSet do
10   newResultSet = newResultSet  $\cup$  manipulate(typeOfResource,p,o);
11   // the function manipulate allows the alteration
12   // based on the semantic information of the retrieved
13   // URIs and evaluates the type of the newly
14   // retrieved resources
15 create triples of the form (resource,p,o) from the newResultSet ;
16 add triples of the form (resource,p,o) to  $S$ ;
17 foreach tuple (p,o) from the newResultSet do
18   if o is a blank node then
19      $S = S \cup$  extract(recursion counter, o, predicateFilter, objectFilter,
20                       literals);
20   else
21      $S = S \cup$  extract(recursion counter -1 ,o, predicateFilter,
22                       objectFilter, literals);
22 return  $S$ 

```

6.2.4 OWL DL Conversion of the Fragment

Extracted knowledge often has to be altered to adhere to OWL DL for further processing, which means explicitly typing classes, properties and objects. Since the knowledge base might not provide (correct) typing information for all individuals, we infer typing information for newly retrieved resources. We follow [Bechhofer and Volz, 2004], who mention an approach, that is based on the idea that if the type of a triple's subject is known, we can infer the type of the object by analyzing the predicate. Since we always start from objects, we possess additional information and therefore are able to extend the rules mentioned in [Bechhofer and Volz, 2004, pp. 673-674]. Given a triple (s, p, o) we can draw the following conclusions:

- If s is an object and p is `rdf:type` then o is a class.
- If s is an object, p is not `rdf:type`, and o not a literal then o is an object.
- If s is a class then o is a class, unless the knowledge source is in OWL Full, in which case we can configure DL-Learner to either ignore such statements or map `rdf:type` (between classes) to `rdfs:subClassOf`. All properties are then ignored except those in the OWL vocabulary having `owl:Class` as range.
- p is an object property if o is a resource and p is a datatype property if o is a literal.

With the help of these observations, we can type all collected resources iteratively, since we know that the starting resources are objects. Thus, we have a consistent way to convert the knowledge fragment to OWL DL based on the information collected during the extraction process. Due to the comparatively small size, deductive reasoning can now be applied efficiently, allowing the application of machine learning techniques.

6.2.5 SPARQL Implementation of Tuple Acquisition

In this section, we will briefly explain how the tuple acquisition interface is implemented efficiently for SPARQL endpoints. The basic triple pattern, which is used, is of the form $\{<resource> ?p ?o\}$ according to the function *acquire(resource)*. The remaining parameters are appended using the FILTER keyword as in the example below. To disburden the SPARQL endpoint, caching is used to remember SPARQL query results which were already retrieved. The extraction algorithm's performance for non-local endpoints is mainly determined by the latency for retrieving SPARQL results via HTTP.

Example 6.3 (Example SPARQL Query on DBpedia)

In this example, we show how we filter out triples using SKOS and DBpedia categories, but leave YAGO classes. Furthermore, links to websites and literals are filtered out.


```

SELECT ?p ?o WHERE {
  <http://dbpedia.org/resource/Angela_Merkel> ?p ?o.
  FILTER (
    !regex(str(?p),
      'http://dbpedia.org/property/website')
    && !regex(str(?p),
      'http://www.w3.org/2004/02/skos/core')
    && !regex(str(?o),
      'http://dbpedia.org/resource/Category')
    && !isLiteral(?o) ). }

```

More optimizations include nested queries according to recursion depth in such a way that it is only necessary to execute one query per example. When retrieving the class hierarchy (*Get all superclasses*) already extracted subclass and other class axioms are remembered and not queried a second time. Because blank nodes in SPARQL result sets do often not relate to the internal blank nodes of knowledge bases (they are iteratively numbered for each result set according to the specification), we use a backtracking technique and assign internal blank node ids.

The implementation of other tuple acquisitors is far simpler. Especially Linked Data can be extracted by just a HTTP request, while the filters are applied after the request. The great advantage of the Linked Data tuple acquisitor is that it allows for cross-boundary acquisition of tuples from different knowledge bases without further configuration and thus enables cross knowledge base accumulation of knowledge.

6.2.6 Usage Scenarios

Instance Data Analysis

We briefly describe two scenarios using GovTrack and MusicBrainz [Swartz, 2002].

Last.fm³⁰ is the worlds largest social music platform. For a given username, we can get information about the last songs a user listened to as RDF³¹. The songs contain ZitGist³² owl:sameAs links, which again refer to MusicBrainz. MusicBrainz is a very large open source music metadata base with plenty of informations about musicians. To obtain a description of the last artists a user has listened to, we pick the MusicBrainz URIs of those artists as positive examples and randomly selected artists as negative ones. To improve the learning process, we converted the MusicBrainz tag cloud into a class hierarchy on the fly by adding a property mapping entry, executed in the *manipulate* function (see Algorithm 6). With the positive examples "Genesis", "Children on Stun", "Robbie Williams",

³⁰<http://www.last.fm/>

³¹via [http://dbtune.org/last-fm/\\$username](http://dbtune.org/last-fm/$username) (description at <http://dbtune.org/last-fm>)

³²<http://www.zitgist.com/>

and "Dusty Springfield", and as negative ones "Madonna", "Cher", and "Dreadzone" we learned the description $\text{UK-Artist} \sqsubset (\text{Rock-Genre} \sqcap \exists \text{bioEvent.Death})$ using OCEL. This gives the user feedback (when expressed in natural language) and allows the system to suggest similar songs, e.g. UK-Rock in this case. As there is a variety of existing media players with MusicBrainz support³³, a learning application could be integrated as plugin into those and employ the Semantic Web to provide descriptions of a users favorite artists, songs, etc.

A similar example for instance analysis can be given for GovTrack, a data set about the US congress containing more than 10 million facts. Amongst other uses, we can apply the presented techniques to learn about the interests and working areas of politicians. To do so, we chose a US senator and queried the GovTrack SPARQL endpoint to return all bills, which were sponsored by him or her. We used this as positive examples and applied DL-Learner. As before with the MusicBrainz tags we performed an enrichment step by converting the subject strings of the bills (financial matter, education) to concepts. We queried the Cyc Foundation browser, which uses OpenCyc³⁴ as background knowledge, to find suitable concepts and integrated them in a hierarchy. As a result, we could see which topics a senator is most interested in and who are cosponsors in bills sponsored by a senator. In this case, the advantage of OCEL is to reduce the often considerable amount of information about a senator to a concise approximate description.

Improving Data Quality

For large knowledge bases, in particular those developed by an Internet community, it is often difficult to maintain a proper classification scheme. A typical example are the DBpedia classification schemata. There have been various attempts to create a classification hierarchy for DBpedia using e.g. the Wikipedia category system as input. Even with good extraction techniques, human errors cannot be completely eliminated and thus articles are assigned to wrong categories or to superfluously many categories. Class learning can be useful in this scenario to learn a complex class C as a possible definition of an existing class A and then verifying whether the instances of C coincide with those of A . Also class expressions can be used to spot data inconsistencies in instance data and to make suggestions for missing instances. In Example 6.4 we show how we can successfully apply the algorithm on DBpedia in different ways to either improve the class schemata, spot inconsistencies in existing categories or make suggestions to Wikipedia editors. Note that a detailed evaluation of the used methods can be found in the next chapter. Here we just evaluated the possibilities for future applications on large knowledge bases.

³³see <http://en.wikipedia.org/wiki/MusicBrainz>

³⁴<http://opencyc.org/>

Wikipedia Categories	Prime Min.	Best Actors	Dyes	Tonga
Total number	71	75	56	50
Correct	53(1)	66(0)	34(0)	50(0)
Incorrect	18(0)	9(3)	22(7)	0(0)
Accuracy	98%	96%	88%	100%

Table 6.7: The table shows a probe of the automatic re-learning method for classes. Sets were evaluated manually, falsely classified individuals in brackets

Example 6.4 (Re-Learning Wikipedia Categories)

We choose 4 Wikipedia categories (Best Actor Academy Award winners, Prime Ministers of the UK, Fluorescent Dyes, Islands of Tonga), which are included in the DBpedia dataset. These categories, as well as the individuals belonging to them, are currently manually maintained by the Wikipedia community, who would benefit greatly from a list of suggestions for missing instances or missing infobox properties. To provide such suggestions a fully automated process is required, when re-learning these categories. While the choice of positive example instances is trivial (all instances assigned to the categories via `skos:subject`), the selection of negative examples is not. If the instances are from a completely different domain or randomly chosen, the correct class expressions are likely to be quite simple. The negative examples were thus obtained by retrieving instances that share the same YAGO classes as the instances in the category. We then randomly selected from this set, such that the number of positive and negative examples were equal. The learning process was then started. The assignment of articles to categories in Wikipedia is not consistent (some categories seem to be confused with tags). The category of British Prime Ministers, for example, also includes instances like *Anthony Eden hat* (a typical hat form worn by Anthony Eden) or *Supermac* (a comic strip about Harold Macmillan). We therefore set 20% as parameter when learning on the fragment. The learned class expressions were used to classify the positive examples in two groups: correctly assigned to the category and incorrectly assigned. We then manually checked these two sets as a Wikipedia editor would do and compared the classification with the information contained in the Wikipedia article.

The results, which are shown in Table 6.7, give a first glance at how useful the generated sets can be for Wikipedia authors. A retrieval of learned concepts (see below for explanation) on DBpedia can further find missing instances.

Since the above described process is fully automated (automatic example choice and concept selection), it can be used to conduct data mining automatically. The retrieved lists could support Wikipedia users, when editing lists and make suggestions about missing entries. The automatically discovered inconsistencies in DBpedia could contribute to future releases of DBpedia itself. \square

SPARQL based Retrieval To validate the results in the described scenario above, we assume that we can retrieve all instances of a learned concept. Usually this is a typical reasoner task. However, as mentioned before, it would be too time-consuming to load the complete knowledge base in a reasoner and pose a retrieval query for the learned concept. A way to solve this problem is to use one or more SPARQL queries to obtain an approximation of the retrieval. We can draw on other work in this area here. The open source project SMART [Battista et al., 2007] implemented a mapping, called DL2SPARQL, to query large knowledge bases. It can be easily tested via their online demonstrator³⁵. Other work in the area of efficient approximate inferences for description logics is also applicable.

Usage for Navigation

Large knowledge bases are very difficult to navigate and explore for end users, in particular in cases with large TBoxes (schema) *and* large ABoxes (instance data). When users search for interesting knowledge with respect to a certain task, they are often able to find interesting objects by searching, browsing or remembering certain objects. However, users usually will not be able to use the full complexity of a knowledge base for posing sophisticated queries corresponding to their enquiries. In these situations class learning combined with fragment extraction can help to suggest high level concepts, thereby allowing the user to gain new insights and explore other relevant objects, which are otherwise hard to find. As an example we choose the DBpedia SPARQL endpoint again, as it is a multi-domain ontology, which could typically be used for research on a certain topic. A user may browse the knowledge through a user interface, which implicitly or explicitly detects some articles, which are relevant for the current enquiry and others which are not. These can be fed into the DL-Learner system (possibly asynchronously called via AJAX in a web application scenario) as positive and negative examples. This was prototypically done in the DBpedia Navigator tool described on page 118. An example is given below:

Example 6.5 (Navigation Use Case)

With the help of class navigation we try to relate certain ancient Greek mathematicians to mathematicians throughout history, that have similarities. Interesting articles are: Pythagoras, Philolaus, Archytas (positive examples)

Uninteresting articles: Socrates, Plato, Zeno of Elea (negative examples)

In this first run we deduce the class *yago:Mathematician* retrieving more than 2000 instances from DBpedia. Those retrieved instances can further be ranked according to certain keywords or rules. We add one of those instances (Democritus) to the negative example set and learn the class expression $Theorist \sqcup (Mathematician \sqcap Physicist)$ in the next run, with which we retrieve slightly above 1000 instances from DBpedia. By adding *Aristoxenus* to the negative examples, the algorithm

³⁵<http://134.117.108.147:8181/smart/query.jsf>

now presents the class expression (among other similar alternatives, which we omitted here) *Believer* \sqcup (*Mathematician* \sqcap *Physicist*). The number of resulting instances from DBpedia shrank to the human manageable size of 159. This list reveals a categorical similarity between the now 8 chosen examples and the instances that belong to the same learned class, containing *Archimedes*, *Aristotle*, *Blaise Pascal*, *Carl Friedrich Gauss*, *Christian Doppler*, *Galileo Galilei*, *Gottfried Leibniz*, *Isaac Newton*, *Leonhard Euler*, *Thales*, just to mention a few famous persons from this list (we might add, that the real value are the not so famous and obvious instances on this list, which are generally harder to identify in a large set of data.). \square

The obtained class expressions mentioned in the example can be converted into natural language and shown as navigation links to the user. Hence, a user interface can present related objects to a user and also tell why they are related.

This concludes the description of the fragment extraction algorithm and we refer to Section 7.4 for an evaluation of it.

6.3 Optimising Coverage Tests

Most of the runtime of a learning algorithm is usually spent for coverage tests, i.e. checking which examples are instances of a hypothesis. There are several reasons for this: The first one is that those tests require instance checks, which can be computationally expensive in OWL. Another reason is that the number of objects to test can be very large, in particular in positive only and class learning problems. We will look closer at those reasons and propose two optimisations: The first optimisation is to use an approximate reasoning procedure. The second optimisation is to perform only as many instance checks as required to be confident that the accuracy of a hypothesis lies within a reasonably small interval.

6.3.1 Approximate and Partial Closed World Reasoning

In the introduced learning problems, we can use a reasoner designed for performing a very high number of instance checks against a knowledge base which can be considered static, i.e. we assume it is not changed during the run of the algorithm. This is reasonable, since the algorithm runtimes are usually in the range of seconds or minutes and users will usually not expect a learning algorithm to react on knowledge base changes during runtime. This allows to optimise the efficiency of reasoning by employing a two step process combining a standard OWL reasoner with a fast approximate instance check method.

We use an approximate incomplete reasoning procedure for instance checks which partially follows a closed world assumption (CWA). First, we use a standard OWL reasoner like Pellet to compute the instances of named classes occurring

in the learning process as well as the property relationships. The obtained inferences, which can be viewed as an ABox, are stored in memory. Afterwards, the reasoning procedure can answer all instance checks (approximately) by using only the inferred knowledge, which results in an order of magnitude speedup compared to using a standard reasoner for instance checks as we will show in Section 7.3.3. The second step, i.e. the instance checks from the inferred knowledge in memory, follows a closed world assumption. This means that we assume the inferred knowledge to be complete with respect to instance checking. Below, we explain why this assumption is useful even though it contradicts the open world assumption in OWL. The procedure is straightforward in that we view the inferred ABox in memory as model \mathcal{I} of the knowledge base and compute whether $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds, which can be done very efficiently. Clearly, considering a single model leads to incomplete reasoning. For instance, it might be the case for a disjunction $C_1 \sqcup C_2$ that some models \mathcal{I} of the knowledge base satisfy $a^{\mathcal{I}} \in C_1^{\mathcal{I}}$ (but not $a^{\mathcal{I}} \in C_2^{\mathcal{I}}$) and all other models satisfy $a^{\mathcal{I}} \in C_2^{\mathcal{I}}$ (but not $a^{\mathcal{I}} \in C_1^{\mathcal{I}}$). In this case, we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$, which cannot be inferred by the described approximate reasoner. See the Oedipus example in [Baader et al., 2007a] for a more detailed description of the need of case distinctions in sound and complete description logic reasoning. While we are aware of this limitation, we accept it due to its rareness and to obtain order of magnitude improvements in performance.

We briefly want to discuss why we are preferring the partial CWA over a straightforward use of a standard OWL reasoner. Note that this has been explained in [Badea and Nienhuys-Cheng, 2000] already, but we repeat the argument here. Consider the following knowledge base containing a person a with two male children a_1 and a_2 :

$$\begin{aligned} \mathcal{K} = \{ & \text{Male} \sqsubseteq \text{Person}, \\ & \text{OnlyMaleChildren}(a), \\ & \text{Person}(a), \text{Male}(a_1), \text{Male}(a_2), \\ & \text{hasChild}(a, a_1), \text{hasChild}(a, a_2) \} \end{aligned}$$

Assume, we want to learn a description for the named class **OnlyMaleChildren**. If we want to compute the score for a concept $C = \text{Person} \sqcap \forall \text{hasChild}.\text{Male}$, describing persons with only male children, we need to check whether a is an instance of C . It turns out that this is true under CWA and not true under OWA³⁶. However, C is a good description of a and could be used to define **OnlyMaleChildren**. For this reason, the CWA is usually preferred in this Machine Learning scenario. Otherwise, universal quantifiers and number restrictions would hardly occur (un-negated) in suggestions for the knowledge engineer – even if their use would be perfectly reasonable. In a broader view, the task of the learning algorithm is to inspect the data actually present, whereas the knowledge engineer can then decide whether these observations hold in general.

³⁶Under OWA, there could be further female children.

6.3.2 Stochastic Coverage Computation

Another optimisation is reducing the number of instance checks. The method will be described for the case of the CELOE score, but similar techniques can be applied to the OCEL and ELTL score computation.

Looking at the definition of acc_c (see Equation 5.1 on page 100) in more detail, we observe that $|R(A)|$ only needs to be computed once, while $|R(A) \cap R(C)|$ and $|R(C)|$ are expensive to compute. However, since acc_c is only a score providing a rough guidance for the learning algorithm, it is usually sufficient to approximate it. Approximating the score value allows us to compute it more efficiently, i.e. test more expressions within a certain time span. However, a too inaccurate approximation can also increase the number of expressions we have to test, since the algorithm is more likely to investigate less relevant areas of the search space. So there is a clear tradeoff in this situation, but, as a rule of thumb, we observed that it is reasonable to approximate accuracy up to $\pm\beta$ (see Definition 5.4 on page 101). In this case, the length penalty balances the inaccuracy of the estimation so that in the worst case the most promising expression is visited later when all promising expressions with short length have been analysed.

The approximation works by testing randomly drawn objects and terminating when we are sufficiently confident that our estimation is within $\pm\beta$. Replacing $|R(A) \cap R(C)|$ with a and $|R(C) \setminus R(A)|$ with b in Equation 5.1, we have to estimate the following expression:

$$acc_c(C, t) = \frac{1}{t+1} \cdot \left(t \cdot \frac{a}{|R(A)|} + \sqrt{\frac{a}{a+b}} \right) \quad (6.1)$$

We first approximate a (or more exactly $t \cdot \frac{a}{|R(A)|}$) by drawing instances of A and testing whether they are also instances of C . To compute a confidence interval efficiently, we use the improved Wald method defined in [Agresti and Coull, 1998]. Assume that we have performed m instance checks where s of them were successful (true), then the 95% confidence interval is as follows:

$$\max(0, p' - 1.96 \cdot \sqrt{\frac{p' \cdot (1-p')}{m+4}}) \text{ to } \min(1, p' + 1.96 \cdot \sqrt{\frac{p' \cdot (1-p')}{m+4}})$$

$$\text{with } p' = \frac{s+2}{m+4}$$

This formula is computationally inexpensive and has been shown to be accurate in [Agresti and Coull, 1998]. Let a_1 and a_2 be the lower and upper border of the confidence interval. We draw instances of A until the interval is smaller than 2β :

$$\frac{a_2}{|R(A)|} - \frac{a_1}{|R(A)|} = \frac{a_2 - a_1}{|R(A)|} \leq 2\beta \quad (6.2)$$

In a top-down learning approach as CELOE, we can discard all expressions with insufficient coverage, since all refinements of those will also have insufficient

coverage ($\frac{a}{|R(A)|}$). What constitutes an insufficient coverage is controlled by the *noise parameter* of the learning algorithm. Noise is a parameter, which specifies the maximum percentage of allowed false negatives for an expression, i.e. the percentage of instances of the given class, which are not instances of the expression (see page 92). If a knowledge base is of very high quality and almost error free, then the noise parameter can be set to a value close to zero. However, if a knowledge base may contain many errors, e.g. knowledge automatically extracted from text, the noise parameter should be set to a high value. The default is 5%, which works well for manually created ontologies.

If the coverage is sufficiently high with respect to the noise parameter, we proceed by approximating b . This is done, similar to a , by subtracting the values at the borders of the confidence interval of b (with respect to Equation 6.1):

$$\frac{1}{t+1} \cdot \left(t \cdot \frac{a_2 - a_1}{|R(A)|} + \sqrt{\frac{a_2}{a_2 + b_1}} - \sqrt{\frac{a_1}{a_1 + b_2}} \right) \leq 2\beta \quad (6.3)$$

This formula also takes the uncertainty in a into account. First note that the overall value of acc_c increases monotonically with larger a and smaller b . So we subtract the value for the lower confidence interval border of a and the upper border of b from the upper border of a and the lower border of b . This is a pessimistic estimate, i.e. the computed interval is overestimated and the approximation is in fact better than $\pm\beta$. Intuitively, the reason is that it is unlikely that both real values of a and b are outside of their computed confidence interval. Also note that it is not hard to show that this approximation process always terminates by analysing the limit $b_1 = b_2$.

Example 6.6 (Stochastic Coverage Test)

Assume we want to learn an equivalence axiom, i.e. $t = 1$, for a class A , where A has 1000 instances and the super classes of A , excluding A itself, have 10000 instances. We choose $\beta = 0.05$. Let C be an expression to test.

First, we approximate a by drawing instances of A and testing whether they are instances of C . After 95 tests of which e.g. 90 were successful, the 95% confidence interval in Equation 6.2 has a width of 0.1006, i.e. is wider than 2β . With another successful test, the confidence interval, which now reaches from 0.8809 to 0.9805, is sufficiently narrow. Hence, we only needed to perform 96 instead of 1000 tests.

We assume that the approximated value is above the noise threshold and continue to compute the overall heuristic value. We do this by drawing instances of super classes of A , excluding A itself, and by testing whether they are instances of C . Equation 6.3 (with $a_1 = 880.9$ and $a_2 = 980.5$) can then be used to estimate whether our approximation is sufficiently accurate. After 64 instance checks of which 32 returned true, the confidence interval is smaller than 2β and we can terminate. In this case, we saved 9936 instance checks. In general, the performance gain is higher for larger knowledge bases. Overall, we get an accuracy value of 67,4%. The expression C covers A well, but is very general. \square

In summary, we presented DBpedia as a typical example of a large knowledge base and introduced a fragment extraction procedure as well as approximate reasoning and coverage tests.

7 Implementation, Evaluation, and Use Cases

In this chapter, we evaluate the introduced methods and show how they can be applied to various problems. The chapter is structured as follows: First, we give an overview over the DL-Learner project, which contains the implementation of the methods presented in this thesis, in Section 7.1. We then evaluate the presented learning algorithms using standard techniques and benchmarks in Section 7.2. The ontology engineering use case, which was one of the motivations of the thesis, is presented in Section 7.3. Section 7.4 provides an evaluation of the fragment extraction approach, which was discussed in Section 6.2. Section 7.5 briefly presents use cases, which are work in progress. Finally, a summarising overview of strengths and limitations of the algorithms according to the evaluation is given in Section 7.6.

7.1 The DL-Learner Project

DL-Learner consists of core functionality, which provides Machine Learning algorithms for solving learning problems in OWL, support for different knowledge base formats, an OWL library, and reasoner interfaces. There are several interfaces for accessing this functionality, a couple of tools which use the DL-Learner algorithms, and a set of convenience scripts.

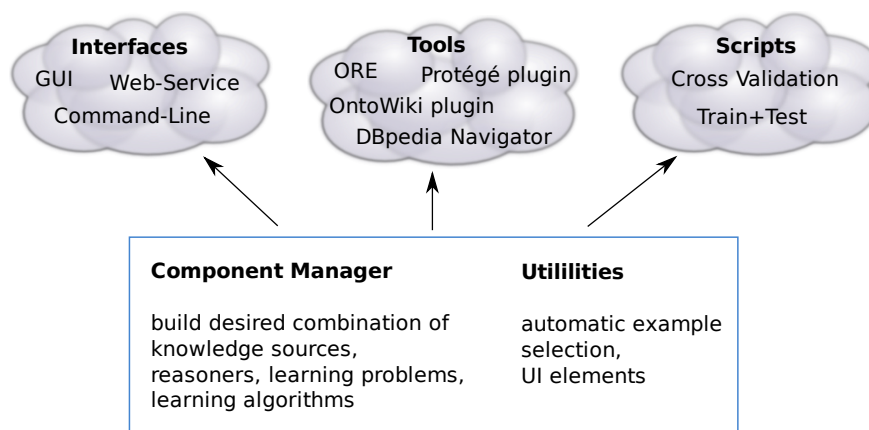


Figure 7.1: Overall structure of the DL-Learner software.

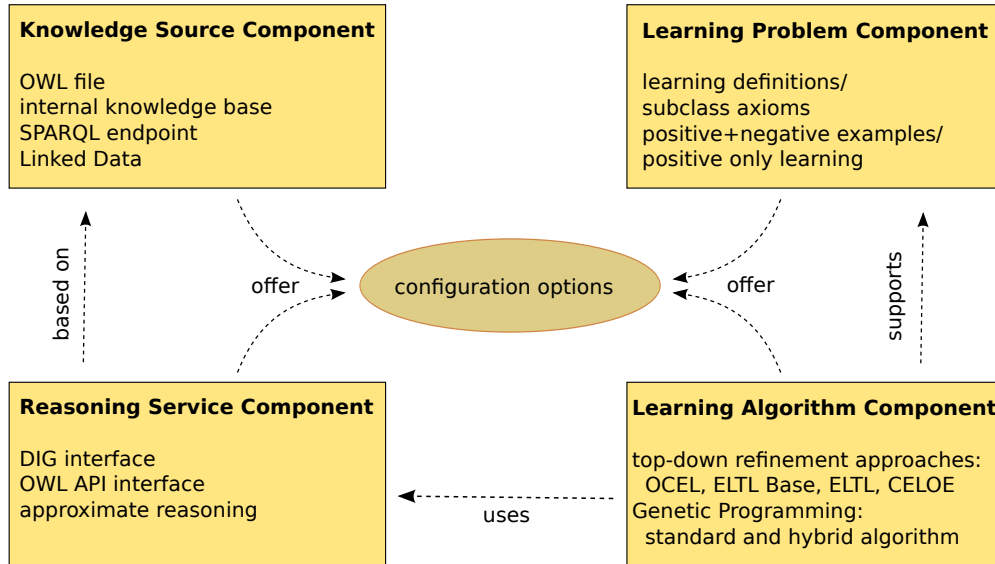


Figure 7.2: The architecture of DL-Learner is based on four component types each of which can have their own configuration options. A component manager can be used to create, combine, and configure components.

The general structure is illustrated in Figure 7.1. To be flexible and easily extensible, DL-Learner uses a component-based model. There are four types of components: knowledge source, reasoning service, learning problem, and learning algorithm. For each type, there are several implemented components and each component can have its own configuration options as illustrated in Figure 7.2. Configuration options can be used to change parameters/settings of a component.

Knowledge Sources integrate background knowledge. Almost all standard OWL formats are supported through the OWL API¹, e.g. RDF/XML, Manchester OWL Syntax, or Turtle. DL-Learner supports the inclusion of several knowledge sources, since knowledge can be widespread in the Semantic Web. In addition, DL-Learner facilitates the extraction of knowledge fragments from SPARQL endpoints and Linked Data. This feature allows DL-Learner to scale up to very large knowledge bases containing millions of axioms as described in Section 6.2.

Reasoner Components provide connections to existing or own reasoners. Two components are the DIG 1.1² and OWL API reasoner interfaces, which allow to connect to all standard OWL reasoners via an HTTP and XML-based mechanism or a Java interface, respectively. Furthermore, DL-Learner offers its own approximate reasoner, described in Section 6.3.1, which uses Pellet³ for bootstrapping and loading the inferred model in memory. Afterwards, instance checks are performed

¹<http://owlapi.sourceforge.net>

²<http://dl.kr.org/dig/>

³<http://clarkparsia.com/pellet/>

very efficiently by using a partial closed world assumption.

Learning Problems specify the problem type, which is to be solved by an algorithm. Currently, the learning problems in Definitions 2.21, 2.22, and 2.23 are implemented. They provide efficient coverage checks as detailed in Section 6.3.

Learning Algorithm components provide methods to solve one or more specified learning problem types. Apart from simple algorithms involving brute force or random guessing techniques, DL-Learner comprises a number of sophisticated algorithms based on hybrid genetic programming with a novel genetic operator [Lehmann, 2007], top-down approaches with refinement operators for the description logic \mathcal{ALC} [Lehmann and Hitzler, 2007c], and the OCEL, ELTL, and CELOE algorithms presented in this thesis.

The homepage of DL-Learner is <http://dl-learner.org> and contains up-to-date information about documentation and development of the software. A manual⁴, which complements the homepage and describes how to run DL-Learner, is included in its release. For developers, the Javadoc of DL-Learner is available online⁵.

The code base of DL-Learner consists of approximately 50,000 lines of code (excluding comments) with its core, i.e. the component framework itself, accounting for roughly 1,500 lines. It is licensed under GPL 3. About 20 learning examples are included in the latest release (to be precise: 132 if smaller variations of existing problems/configurations are counted). 27 unit tests based on the JUnit framework are used to detect errors.

There are several interfaces available to access DL-Learner: To use components programmatically, the core package, in particular the component manager, can be of service. Similar methods are also available at the web service interface, which is based on WSDL. DL-Learner starts a web service included in Java 6, i.e. no further tools are necessary. For end users, a command line interface is available. Settings are stored in *conf files*, which can then be executed in a similar fashion to other ILP tools. A prototypical graphical user interface is equally available, which can create, load, and save conf files. It provides widgets for modifying components and configuration options. An advantage of the component-based architecture is that all the interfaces mentioned need not to be changed, when new components are added or existing ones modified. This makes DL-Learner easily extensible. Another means to access DL-Learner, in particular for ontology engineering, is through plugins for the ontology editors OntoWiki⁶ and Protégé⁷. The OntoWiki plugin is under construction, but can be used in its latest repository version. The

⁴<http://dl-learner.org/files/dl-learner-manual.pdf>

⁵<http://dl-learner.org/javadoc>

⁶<http://ontowiki.net>

⁷<http://protege.stanford.edu>

Protégé 4 plugin is included in the official Protégé plugin repository, i.e. it is easy to install within Protégé. More details on the two plugins follow in Section 7.3.

7.2 ILP Learning Problems

To assess the performance of the described learning algorithms on real problems and benchmarks, they were compared to other machine learning tools. The evaluation process was performed in two steps: First, a comparison with algorithms in description logics was performed. Afterwards, a comparison with state-of-the-art ILP tools on a challenging problem was made.

7.2.1 Comparison with other Algorithms based on Description Logics

Doubtless, there is a lack of evaluation standards in ontology learning from examples. In order to overcome this problem, we converted the background knowledge of several existing learning problems to OWL ontologies. Besides the train problem (described on page 94), we also investigated the problems of learning family relationships from FORTE [Richards and Mooney, 1995], learning poker hands, and understanding the moral reasoning of humans. The two latter examples were taken from the UCI Machine Learning repository⁸. For the poker example, we defined two goals: learning the definition of a pair and of a straight. Similarly, the moral reasoner examples were divided into two learning tasks: the original one, where the intended solution is quite short, and a problem where we removed an important intermediate concept, such that the smallest possible solution became more complex. For the FORTE family data set, we defined the problem of learning the definition of an uncle (originally defined in [Lehmann, 2007]), where one possible solution is:

$$\text{Male} \sqcap (\exists \text{sibling}.\exists \text{parent}.\top \sqcup \exists \text{married}.\exists \text{sibling}.\exists \text{parent}.\top)$$

The poker example has medium size, but is not very complex with respect to its terminology. The moral reasoner, however, is an expressive ontology, which we derived from a theory given as a logic program. We designed the FORTE problem to be slightly more difficult for our algorithm, such that no simple and short solutions exist. Overall, the solutions of the examples cover a range of different concept constructors and are of varying length and complexity. Note that in the next section, we will describe a current ILP benchmark, the carcinogenesis problem, in order to compare the described algorithm with inductive learning algorithms which are not based on description logics.

⁸<http://www.ics.uci.edu/~mlearn/MLRepository.html>

As a reasoner we used Pellet⁹, which was connected to the learner via the DIG reasoner interface¹⁰ for YinYang [Iannone et al., 2007] and the OWL API¹¹ interface for DL-Learner (because DIG does not support asking for domains and ranges), on a 2.2 GHz Dual CPU machine with 2 GB RAM. We compared our results with those from YinYang and other algorithms we have implemented within the DL-Learner framework. In particular, we compared with a hybrid algorithm using so called genetic refinement operators [Lehmann, 2007], i.e. adapted refinement operators within a Genetic Programming (GP) framework. For reference we also compared with a standard GP learning algorithm, which has been applied to the learning problem in description logics and is also included in DL-Learner. For all algorithms, we used five fold cross validation. We are not aware of other systems available for comparison. The system in [Cohen and Hirsh, 1994] is no longer available and the approach in [Badea and Nienhuys-Cheng, 2000] was not fully implemented. DL-FOIL [Fanizzi et al., 2008] is not publicly available yet. We compare the algorithms against OCEL and ELTL.

For YinYang we used the same settings as in all examples included in its release and for OCEL and ELTL we used the standard settings. For the GP algorithms, we chose a fixed number of 50 generations with a population of 500 individuals. A generational algorithm with rank selection and activated elitism was used. The algorithms were initialised using the ramped-half-and-half method with maximum depth 6. For the standard GP algorithm, a crossover probability of 80 percent and 2 percent mutation probability was set. The hybrid approach was configured to 65 percent genetic refinement, 20 percent crossover, and 2 percent mutation. In both cases, the fitness measure parameter α was adjusted to a low value (0.002), such that the correct solutions have a sufficiently high value in the GP fitness function. The settings are similar to those found to be suitable in [Lehmann, 2007], where population size is varied between 100 and 700. Naturally, a GP algorithm always allows to increase the number of invested resources and varying the population size is one of the ways to do this. We finally picked a population size of 500 for our evaluation, since in example runs it seemed to deliver a good tradeoff between runtime and cross validation accuracy. Overall, the runtime of the GP algorithms is often much higher than those of DL-Learner and YinYang in this setting.

Tables 7.1 and 7.2 summarise the results we obtained. As a statistical significance test, we used a t-test with 95% confidence interval. In all cases, OCEL was able to learn a correct definition on the training set, which in most cases also was correct on the testing set. For ELTL, the target language was sometimes not sufficiently expressive, i.e. did not allow to construct good solutions. YinYang could not handle the second poker problem (it produces an error after trying to compute most specific concepts). Similarly, the FORTE problem could only be handled after removing certain examples (4 out of 86). Figure 7.3 visualises the

⁹<http://pellet.owldl.com>

¹⁰<http://dl.kr.org/dig/>

¹¹<http://owlapi.sf.net/>

problem	DL-Learner OCEL			
	time (s)	time 2 (s)	length	correct (%)
trains	0.8 ± 0.3	0.1 ± 0.0	5.0 ± 0.0	100.0 ± 0.0
moral I	2.8 ± 0.3	0.1 ± 0.0	8.0 ± 0.0	97.8 ± 5.0
moral II	2.7 ± 0.4	0.1 ± 0.0	8.0 ± 0.0	97.8 ± 5.0
poker I	3.4 ± 0.1	0.0 ± 0.0	5.0 ± 0.0	100.0 ± 0.0
poker II	19.7 ± 10.6	1.5 ± 0.1	11.0 ± 0.0	100.0 ± 0.0
forte	13.4 ± 1.8	0.3 ± 0.1	13.4 ± 0.9	98.9 ± 2.5

problem	DL-Learner ELTL Base		
	time (s)	length	correct (%)
trains	1.2 ± 0.2	9.8 ± 2.7	100.0 ± 0.0
moral I	0.7 ± 0.1	2.6 ± 0.9	74.2 ± 6.6
moral II	0.7 ± 0.0	2.6 ± 0.9	74.2 ± 6.6
poker I	0.8 ± 0.0	7.0 ± 0.0	100.0 ± 0.0
poker II	1.2 ± 0.6	11.5 ± 1.0	98.1 ± 3.8
forte	0.0 ± 0.0	1.0 ± 0.0	73.1 ± 10.7

problem	DL-Learner ELTL		
	time (s)	length	correct (%)
trains	1.4 ± 0.6	5.8 ± 1.8	90.0 ± 22.4
moral I	2.2 ± 0.6	3.0 ± 0.0	88.6 ± 19.3
moral II	2.1 ± 0.6	7.0 ± 2.8	81.7 ± 17.1
poker I	1.2 ± 0.1	5.0 ± 0.0	100.0 ± 0.0
poker II	1.3 ± 0.1	12.0 ± 2.0	100.0 ± 0.0
forte	2.3 ± 0.8	13.8 ± 7.2	89.3 ± 8.3

Table 7.1: Evaluation results for the OCEL and ELTL algorithms.

problem	YinYang		
	time (s)	length	correct (%)
trains	0.2 ± 0.1	8.1 ± 1.5	100.0 ± 0.0
moral I	28.8 ± 12.1	69.0 ± 16.1	50.0 ± 21.7
moral II	32.6 ± 9.5	70.7 ± 21.8	62.5 ± 28.0
poker I	7.2 ± 0.8	43.2 ± 12.1	100.0 ± 0.0
poker II	-	-	-
forte	26.4 ± 9.4	22.1 ± 12.0	90.0 ± 5.6

problem	DL-Learner GP		
	time (s)	length	correct (%)
trains	33.4 ± 3.1	3.4 ± 0.9	60.0 ± 41.8
moral I	28.9 ± 1.4	1.0 ± 0.0	86.1 ± 10.0
moral II	31.0 ± 5.9	1.6 ± 0.9	62.8 ± 9.1
poker I	465.9 ± 261.3	3.4 ± 2.2	84.0 ± 21.9
poker II	283.3 ± 12.7	1.0 ± 0.0	92.7 ± 0.3
forte	235.9 ± 74.6	3.0 ± 0.0	88.1 ± 8.0

problem	DL-Learner hybrid GP		
	time (s)	length	correct (%)
trains	10.5 ± 1.1	4.6 ± 0.9	80.0 ± 44.7
moral I	139.9 ± 10.0	3.0 ± 0.0	100 ± 0.0
moral II	118.7 ± 26.8	2.8 ± 1.1	71.9 ± 13.1
poker I	709.1 ± 60.7	5.0 ± 0.0	100.0 ± 0.0
poker II	1054.1 ± 36.5	1.0 ± 0.0	92.7 ± 0.3
forte	285.0 ± 25.6	3.0 ± 0.0	88.1 ± 8.0

Table 7.2: Results for YinYang and Genetic Programming approaches.

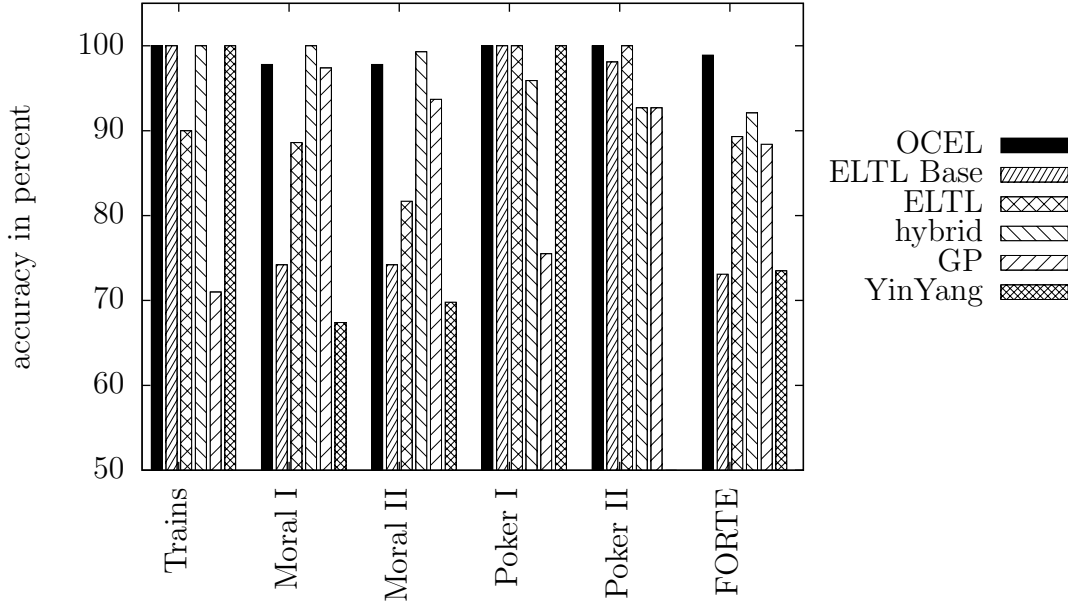


Figure 7.3: Accuracy comparison: OCEL has a statistically significantly higher accuracy than all others on the complex moral reasoner, poker II, and forte problems, while none of the other algorithms performs statistically significantly better on any of the other learning problems.

obtained cross validation accuracies. OCEL has a statistically significantly higher accuracy than the GP and YinYang approaches on the complex moral reasoner, poker II, and forte problems, while none of the other algorithms performs statistically significantly better on any of the other learning problems. The hybrid GP approach generally performed at least as good as the standard GP approach and often it was (statistically significantly) better.

As another interesting criterion, we recorded the length of learned concepts (see Figure 7.4). A notable observation is that YinYang produces longer concepts than OCEL and ELTL with high statistic significance. In most cases, this rendered those concepts hard to read for humans, which is a disadvantage for symbolic classifiers. In contrast, the genetic programming approaches often produce short concepts. The standard GP algorithm usually learned such short solutions, because it is often unable to find more accurate longer definitions. The hybrid approach is more likely to find complex definitions, e.g. it sometimes found one of the possible solutions ($\text{Severity_harm} \sqcap \neg \text{Benefit_victim} \sqcap (\text{Responsible} \sqcup \text{Vicarious})$) of the second moral reasoner problem.

Figure 7.5 compares the runtime of the different algorithms (note the logarithmic scale). Overall, the GP algorithms have a higher runtime. As mentioned before, they can be parametrised to have a shorter runtime if we are willing to accept a decline in accuracy. In general, OCEL and ELTL are statistically signif-

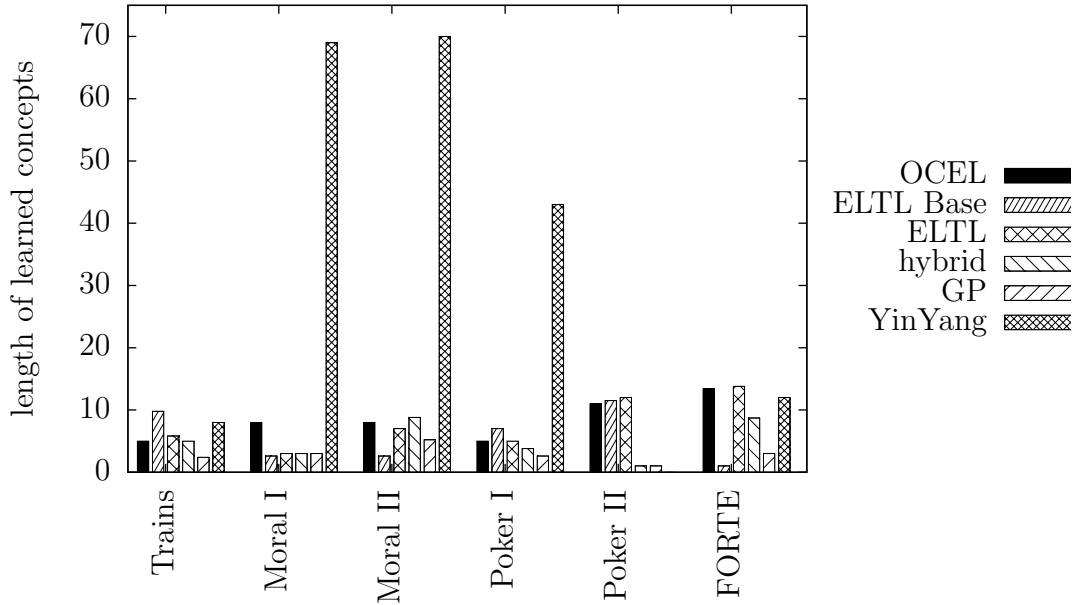


Figure 7.4: Comparison of the length of the learned concepts: YinYang produces longer concepts, which were in general more difficult to read for humans, than OCEL with high statistic significance. The genetic programming approaches, in particular the standard variant, are often unable to solve the more complex problems in reasonable time.

icantly faster than all other approaches for all problems except trains.

For OCEL, we included another column "time 2" in Table 7.2. These are the learning times when employing the built-in approximate OWL reasoner instead of pure Pellet (see Section 6.3.1 for a description).

Although partially using closed world reasoning, as done in this approach, can change the node score and therefore influences the learning process, it did not impact on the cross validation accuracy in the presented examples except for a single fold in the poker II learning problem, where it stopped after finding a shorter concept covering all training set examples, and therefore had an overall cross validation accuracy of 96.2% instead of the 100% shown in the table.

All learning problems are available at the DL-Learner subversion repository¹² and a script is provided to reproduce the results presented here.

7.2.2 Comparison with other ILP approaches

To compare OCEL with other ILP approaches and to apply it in a realistic scenario, we choose the problem of predicting carcinogenesis. The aim of this task

¹²Browsable e.g. via <http://dl-learner.svn.sourceforge.net/viewvc/dl-learner/>.

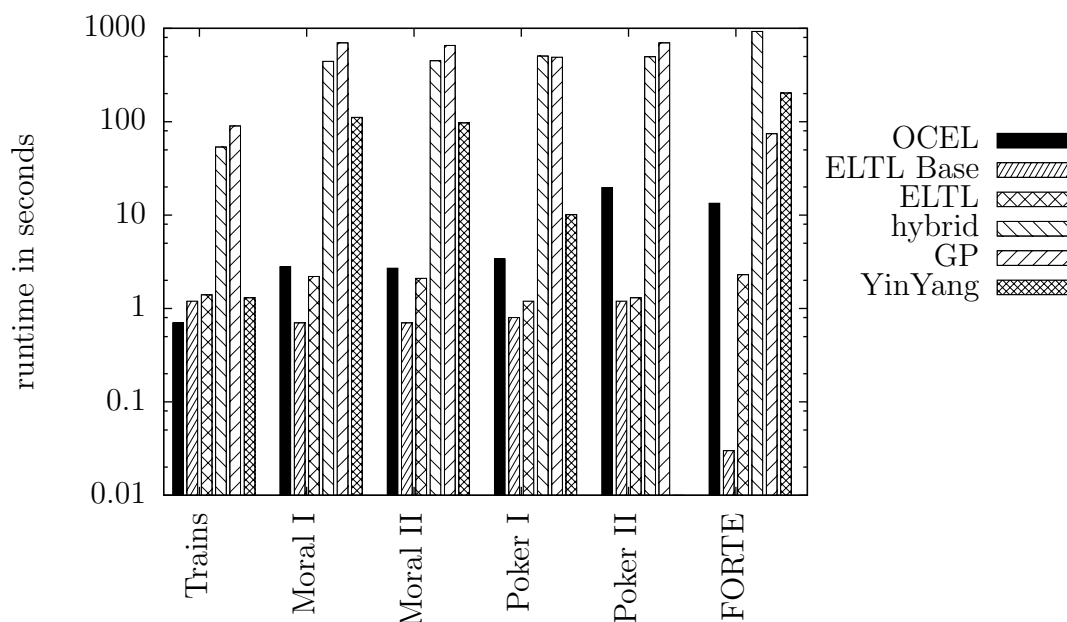


Figure 7.5: Runtime comparison: The GP algorithms generally have the highest runtimes unless we trade accuracy for runtime. OCEL and ELTL are statistically significantly faster than all other approaches for all problems except trains.

is to predict whether a chemical compound causes cancer given its structure and the results of bio-assays. This has been recognised as an important research area: "Obtaining accurate structural alerts for the causes of chemical cancers is a problem of great scientific and humanitarian value" [Srinivasan et al., 1997]. Although this is one of the most well-researched problems in Machine Learning, it is still important and challenging.

One of the problems we have to face when benchmarking DL-Learner is, of course, that many of the problems are available in a Prolog-like syntax. The first step to apply DL-Learner to the carcinogenesis problem was again to convert the original data¹³ into an OWL ontology. To do this, we extended DL-Learner with a Prolog parser and wrote a mapping script to convert the carcinogenesis files into OWL. OWL (description logics) and logic programs have incomparable expressivity. It is sometimes impossible and often not trivial to convert between both representations. For carcinogenesis such a mapping is possible, but required at least a superficial understanding of the domain. The mapping script we used and the resulting ontology are both freely available at the DL-Learner download page.¹⁴ During the transformation process almost no knowledge was lost or added. The resulting ontology contains 142 atomic concepts, 19 roles and datatypes, 22373

¹³<http://web2.comlab.ox.ac.uk/oucl/research/areas/machlearn/cancer.html>

¹⁴<http://sourceforge.net/projects/dl-learner>

objects, and more than 74000 facts.

We used the approximate reasoner introduced in Section 6.3.1. Standard OWL reasoners turned out to be much too slow in answering reasoning requests for this ontology. Furthermore, we also enabled all extensions presented in Section 4.1.4. Again, all tests were run on a 2.2 Ghz Dual core machine with 4 GB RAM.

The most sensible parameter of OCEL in the case of carcinogenesis prediction is noise (bounding the minimum acceptable training set accuracy of the learned definition). To find an appropriate setting for the noise parameter, we divided the available examples into two sets. In a first phase, 30% of the 337 examples were used to find the noise parameter value. This was done by starting from a noise value of 50% and descending in one percent steps. We measured the ten fold cross validation accuracy for each of those values. It turned out that a noise value of 30% has the highest accuracy (71.8%). In a second phase, we used this parameter to measure the ten fold cross validation accuracy on the second set containing 70% of all examples. This lead to an accuracy of 67.7% as shown in Table 7.3 together with results of other approaches using the same background knowledge (many of those use Aleph, a state-of-the-art Inductive Logic Programming system, as their basis).

approach/tool	accuracy	readability	reference
OCEL	67.7% \pm 11.3%	+	
Aleph Ensembles	59.0% to 64.5%	–	[Dutra et al., 2003]
Boosted Weak ILP	61.1%	–	[Jiang and Colton, 2006]
Weak ILP	58.7%	–	[Jiang and Colton, 2006]
Aleph DTD 0.7	57.9% \pm 9.8%	o	[Železný et al., 2003]
Aleph RRR 0.9	57.6% \pm 6.4%	o	[Železný et al., 2003]
Aleph DTD 0.9	56.2% \pm 9.0%	o	[Železný et al., 2003]
Aleph RRR 0.7	54.8% \pm 9.0%	o	[Železný et al., 2003]

Table 7.3: Overview of the accuracy of different ILP approaches applied to the carcinogenesis prediction problem. Many of those are improvements or different settings of Aleph. Accuracy and standard deviation refer to values obtained through 10 fold cross validation. Legend: DTD = Deterministic Top Down, RRR = Randomized Rapid Restarts, +, o, – stand for good, moderate, and bad readability of learning results (see text)

For all approaches, where the standard deviation was given in the articles, we calculated whether the difference in accuracy is statistically significant using a t-test. We obtained P values of 0.0508 vs. Aleph DTD 0.7, 0.0231 vs. Aleph RRR 0.9, 0.0206 vs. Aleph DTD 0.9, and 0.0107 vs. Aleph RRR 0.7. Values ≤ 0.05 (3 out of 4 in this case) are statistically significant with a standard confidence interval of 95%.

The average runtime for a noise value of 30% was 950 seconds with a concept length of 20 when measured on all examples. We will briefly compare this with the other approaches in Table 7.3.

[Dutra et al., 2003] is a bagging approach combining 1 to 100 hypotheses generated by Aleph. The results vary from 59.0% to 64.5% accuracy depending on the chosen ensemble size. We believe that a combination of a high number of hypotheses scores lower on human interpretability, in particular since the concepts provided by OCEL are quite compact. The results themselves were computed on Condor¹⁵, a high throughput computing system and consumed 53580 CPU hours including 3 other experiments apart from carcinogenesis. Even taking the different ensemble sizes and parameter optimisation phases in this experiment into account, our approach seems to be competitive.

[Jiang and Colton, 2006] did not record the runtime of experiments. The system uses a boosting approach with 50 to 100 base classifiers based on WeakILP. Using a similar length measure than the one we defined for DL concepts, i.e. counting all logical symbols, the summed length is approximately 1000 compared to 13.4 in our case. Hence, we also consider this approach to produce less readable results.

[Železný et al., 2003] reports runtimes of 6 to 7 hours for the two DTD approaches on a 1.5 GHz machine with 512 MB RAM. The RRR approaches are much faster and need only 26 minutes. Regarding readability, the length of learned programs is about 100. Hence, they are much shorter than those of the two other approaches, but can still be considered harder to interpret than the results of DL-Learner.

To illustrate the influence of the noise parameter, we additionally measured ten fold cross validation accuracy, runtime, and concept length on all examples with different noise values. The results are shown in Table 7.4. Since the noise parameter acts as a termination criterion, we observe, as expected, that lower noise values lead to significant increases in runtime. It becomes increasingly computationally expensive for the learning algorithm to find concepts satisfying the termination criterion and those concepts are usually also more complex as evident from the last column of the table. Therefore, setting the noise values too low can also lead to learning unnecessary complex concepts as shown in Figure 7.6.

As an example of a learned concept, the following definition was one of the more complex concepts learned with noise=28%:

$$\begin{aligned} &(\text{Compound} \sqcap \neg \exists \text{hasAtom.}(\text{Nitrogen-35} \sqcup \text{Phosphorus-60} \\ &\quad \sqcup \text{Phosphorus-61} \sqcup \text{Titanium-134}) \\ &\sqcap (\geq 3 \text{hasStructure.}(\text{Halide} \sqcap \neg \text{Halide10}) \\ &\quad \sqcup (\text{amesTestPositive} = \text{true} \sqcap \geq 5 \text{hasBond.}(\neg \text{Bond-7}))) \end{aligned}$$

This can be phrased in natural language as:

¹⁵<http://www.cs.wisc.edu/condor/>

noise(%)	accuracy(%)	runtime(s)		length
40	62.9 \pm 8.6	0.6 \pm	0.6	4.9 \pm 1.4
39	62.9 \pm 8.6	0.6 \pm	0.7	4.9 \pm 1.4
38	65.9 \pm 8.3	1.5 \pm	0.2	7.0 \pm 0.0
37	65.9 \pm 8.3	5.2 \pm	7.8	7.6 \pm 1.3
36	65.9 \pm 8.3	6.9 \pm	8.5	7.9 \pm 1.4
35	65.9 \pm 8.3	12.7 \pm	9.6	8.8 \pm 1.6
34	64.4 \pm 6.6	31.6 \pm	25.7	9.7 \pm 0.7
33	64.7 \pm 6.5	73.6 \pm	88.4	9.8 \pm 0.9
32	67.4 \pm 7.9	160.0 \pm	197.2	10.7 \pm 3.1
31	66.4 \pm 7.5	426.9 \pm	324.2	14.1 \pm 3.7
30	65.9 \pm 8.7	843.5 \pm	538.1	17.9 \pm 4.5
29	66.8 \pm 8.1	1613.9 \pm	922.3	23.2 \pm 5.0
28	66.5 \pm 9.0	3158.3 \pm	1680.8	29.6 \pm 5.8

Table 7.4: The influence of the noise parameter on ten fold cross validation accuracy, runtime, and length of learned concepts. We see that for lower noise values it becomes increasingly hard to satisfy the termination criterion (hence the increase in runtime) and the learned concepts are longer and more complex. The maximum accuracy is reached for 32% noise and stays on a similar level afterwards, which indicates that the additional structures in those longer concepts do not greatly affect classification results. Note that the runtime does not include the time to load the knowledge base into the reasoner and prepare it, which takes additional 36 seconds on our test machine.

*A chemical compound is carcinogenic iff ...
...it does not contain a Nitrogen-35, Phosphorus-60, Phosphorus-61,
or Titanium-134 atom
...and it has at least three Halide – excluding Halide10 – structures
or the ames test of the compound is positive and there are
at least five atom bonds which are not of bond type 7.*

Overall, OCEL is able to learn accurate and short concepts with a reasonably low number of expensive reasoner requests. Note that it learns in an expressive language with arbitrarily nested structures, as can be seen in the concept above. Learning many levels of structure has recently been identified as a key issue for structured Machine Learning [Dietterich et al., 2008], and our work provides a clear advance on this front.

The evaluations show that our approach is competitive with state-of-the-art ILP systems when the approximate reasoning technique is used.

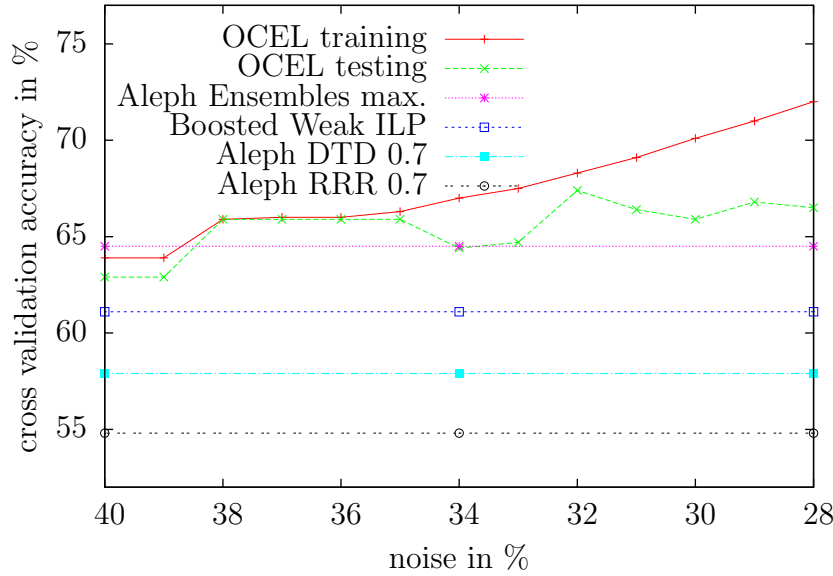


Figure 7.6: Illustration of ten fold cross validation accuracies of OCEL on the carcinogenesis benchmark for different noise values. For lower noise values the difference between training and testing accuracy (averaged over all ten folds) becomes larger, which can be interpreted as a sign of overfitting, i.e. the learned concepts are unnecessarily complex. As a reference, we included the accuracies other tools in Table 7.3 as horizontal lines.

7.3 Ontology Engineering

The Semantic Web has recently seen a rise in the availability and usage of knowledge bases, as can be observed within the Linking Open Data Initiative or the TONES¹⁶ and Protégé¹⁷ ontology repositories as well as the Sindice Semantic Web index and the Watson Semantic web search engine¹⁸. Despite this growth, there is still a lack of knowledge bases that consist of sophisticated schema information and instance data adhering to this schema. Several knowledge bases, e.g. in the life sciences, only consist of schema information, while others are, to a large extent, a collection of facts without a clear structure, e.g. information extracted from data bases or texts. Combining both allows powerful reasoning, consistency checking, and improved querying possibilities. Being able to learn OWL class expressions could be a step towards achieving this goal (see Example 1.1).

We argue that the approach and plugins presented here are the first ones to be practically usable by knowledge engineers for learning class expressions. Using machine learning for the generation of suggestions instead of entering them

¹⁶<http://owl.cs.manchester.ac.uk/repository/>

¹⁷http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library

¹⁸<http://watson.kmi.open.ac.uk>

manually has the advantage that 1.) the given suggestions fit the instance data, i.e. schema and instances are developed in concordance, and 2.) the entrance barrier for knowledge engineers is significantly lower, since understanding an OWL class expression is easier than manually analysing the structure of the knowledge base and creating a class expression manually. A disadvantage is that the method only works when instance data (an ABox) is available. Hence, it is especially useful in those cases, where an ontological schema is created from existing instance data, i.e. a so called “schema last” or “grassroots” approach is used, or where an existing knowledge base should be maintained.

The section is structured as follows: We present two plugins for knowledge engineers for the popular Protégé and OntoWiki ontology editors based on CELOE. Afterwards, we evaluate the CELOE algorithm on several knowledge bases and draw conclusions.

7.3.1 The Protégé Plugin

After implementing and testing CELOE, we integrated it in Protégé and OntoWiki. Together with the Protégé 4 developers, we extended their plugin mechanism to be able to seamlessly integrate the DL-Learner plugin as an additional method to create class expressions. This means that the knowledge engineer can use the algorithm exactly where it is needed without any additional configuration steps. The plugin has also become part of the official Protégé 4 repository, i.e. it can be directly installed from within Protégé.

A screenshot of the plugin is depicted in Figure 7.7. To use the plugin, the knowledge engineer is only required to press a button, which then starts a new thread in the background. This thread executes the learning algorithm. CELOE is an anytime algorithm, i.e. at each point in time we can always see the currently best suggestions. The GUI updates the suggestion list each second until the maximum runtime – 10 seconds per default – is reached. This means that the perceived runtime, i.e. the time after which only minor updates occur in the suggestion list, is often only one or two seconds for small ontologies. For each suggestion, the plugin displays its accuracy.

When clicking on a suggestion, it is visualized by displaying two circles: One stands for the instances of the class to describe and another circle for the instances of the suggested class expression. Ideally, both circles overlap completely, but in practice this will often not be the case. Clicking on the plus symbol in each circle shows its list of individuals. Those individuals are also presented as points in the circles and moving the mouse over such a point shows information about the individual. Red points show potential problems, where it is important to note that we use a closed world assumption to detect those. For instance, in our initial example in Chapter 1, a capital which is not related via the property `isCapitalOf` to an instance of `Country` is displayed as red dot. If there is not only a potential problem, but adding the expression would render the ontology inconsistent, the suggestion itself is marked red and a warning message is displayed. Please note

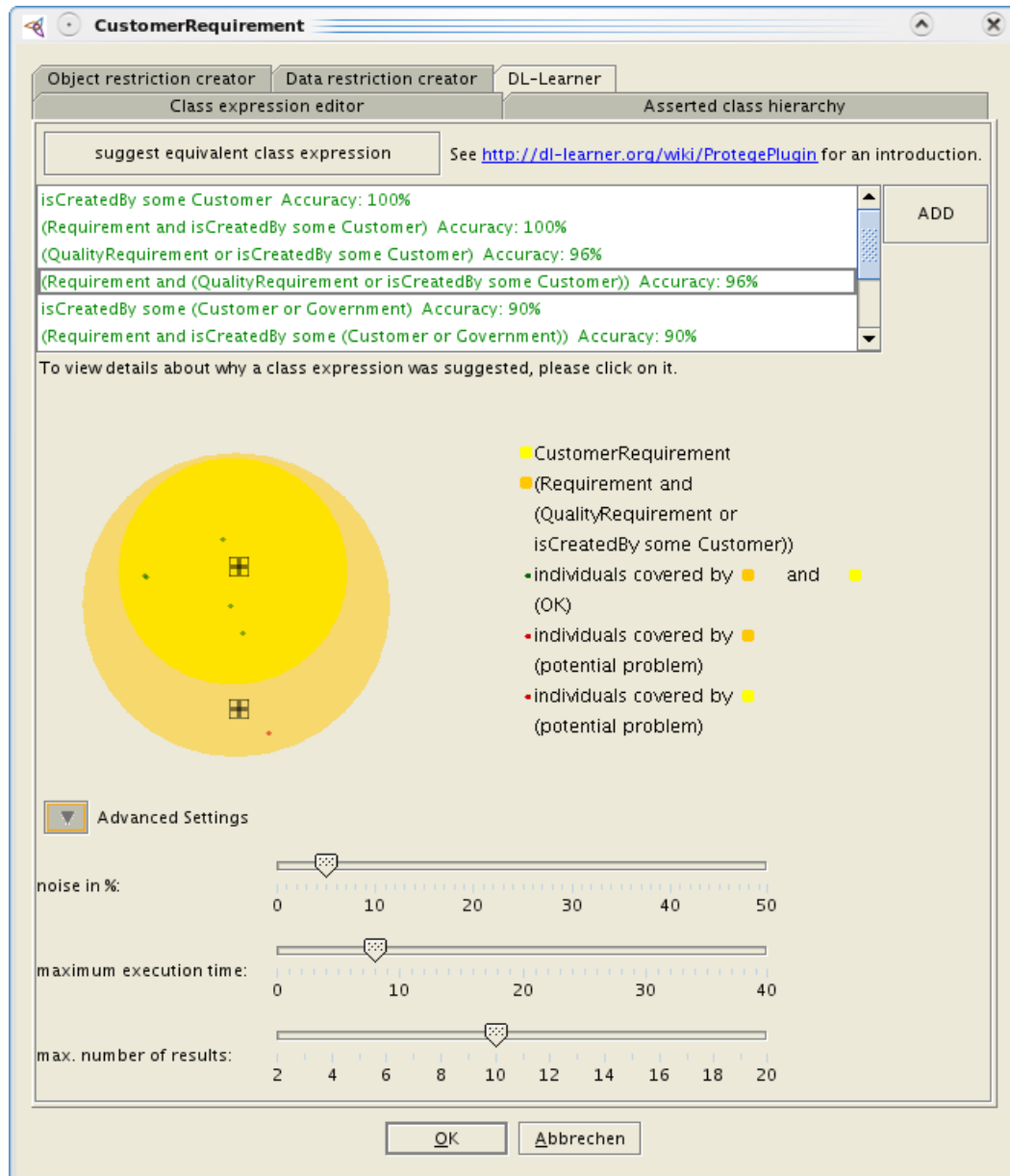


Figure 7.7: A screenshot of the DL-Learner Protégé plugin. The tabs at the top present various possibilities to create class expressions offered by Protégé. The plugin integrates seamlessly with them. The user is only required to press the “suggest equivalent class expressions” button and within a few seconds they will be displayed as list with the most promising ones on top. If desired, the knowledge engineer can visualize the instances of the expression to detect potential problems. At the bottom, optional expert settings can be made to configure the learning algorithm.

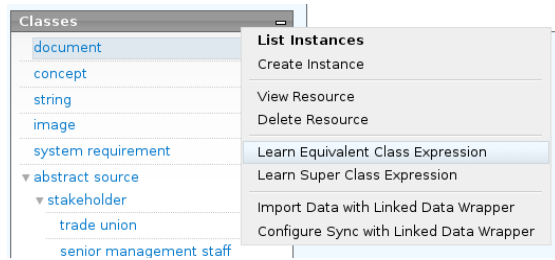


Figure 7.8: The DL-Learner plugin can be invoked from the context menu of a class in OntoWiki.

that accepting such a suggestion can still be a good choice, because the problem often lies elsewhere in the knowledge base, but was not obvious before, since the ontology was not sufficiently expressive for reasoners to detect it. The plugin homepage¹⁹ shows a screencast, where the ontology becomes inconsistent after adding the axiom, and the real source of the problem is fixed afterwards. Being able to make such suggestions can be seen as a strength of the plugin.

The plugin allows the knowledge engineer to change expert settings, e.g. the noise parameter introduced on page 92. For instance, a user may have erroneously added that Sydney is a capital. CELOE is designed to handle noise and the visualisation of the suggestions will reveal those false class assignments. If the knowledge engineer deals with very noisy ontologies, she can choose to increase the default value of 5%. Further options include the maximum suggestion search time and the number of returned results. In future work, we may also allow to fine-tune the target language, i.e. which OWL language constructs can be used in class expressions. This is useful in cases where the ontology should remain within an OWL 2 profile, i.e. OWL 2 EL.

7.3.2 The OntoWiki Plugin

Analogous to Protégé, we created a plugin for OntoWiki [Auer et al., 2006b]. OntoWiki is a lightweight ontology editor, which allows distributed and collaborative editing of knowledge bases. It focuses on wiki-like, simple and intuitive authoring of semantic content, e.g. through inline editing of RDF content, and provides different views on instance data.

Recently, a fine-grained plugin mechanism and extension architecture was added to OntoWiki. The DL-Learner plugin is technically realised by implementing an OntoWiki component, which contains the core functionality, and a module, which implements the embedding in the user interface. The DL-Learner plugin can be invoked from several places in OntoWiki, for instance through the context menu of classes as shown in Figure 7.8.

The plugin accesses DL-Learner functionality through its WSDL-based web service interface. Jar files containing all necessary libraries are provided in the

¹⁹<http://dl-learner.org/wiki/ProtegePlugin>

plugin. If a user invokes the plugin, it scans whether the web service is online at its default address. If not, it is started automatically.

A major technical difference compared to the Protégé plugin is that the knowledge base is accessed via SPARQL, since OntoWiki is a SPARQL-based web application. In Protégé, the current state of the knowledge base is stored in memory in a Java object. As a result, we cannot easily apply a reasoner on an OntoWiki knowledge base. To overcome this problem, we use the DL-Learner fragment selection mechanism described in Section 6.2. The fragment selection is only performed for medium to large-sized knowledge bases. Small knowledge bases are retrieved completely and loaded into the reasoner. While the fragment selection can cause a delay of several seconds before the learning algorithm starts, it also offers flexibility and scalability. For instance, we can learn class expressions in large knowledge bases such as DBpedia in OntoWiki²⁰.

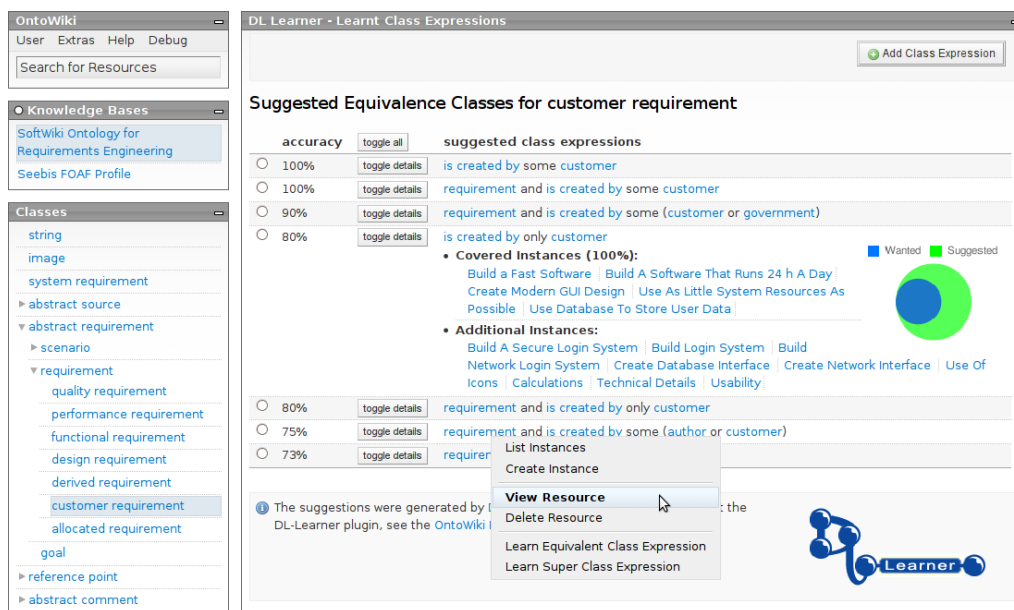


Figure 7.9: Screenshot of result table of the DL-Learner plugin in OntoWiki.

Figure 7.9 shows a screenshot of the OntoWiki plugin applied to the SWORE ontology [Riechert et al., 2007b]. Suggestions for learning the class “customer requirement” are shown in Manchester OWL Syntax. Similar to the Protégé plugin, the user is presented a table of suggestions along with their accuracy value. Additional details about the instances of “customer requirement”, covered by a suggested class expressions and additional instances covered can be viewed via a toggle button. The modular design of OntoWiki allows rich user interaction: Each resource, e.g. a class, property, or individual, can be viewed and subsequently modified directly from the result table as shown in the screenshot. For instance,

²⁰OntoWiki is currently undergoing an extensive development, aiming to support handling such large knowledge bases. A release supporting this is scheduled for mid 2010.

a knowledge engineer could decide to import additional information available as Linked Data and run the CELOE algorithm again to see whether different suggestions are provided with additional background knowledge.

7.3.3 Evaluation of CELOE

To evaluate the suggestions made by CELOE, we tested it on a variety of real world ontologies of different sizes and domains. The goals of the evaluation are to

1. determine whether the described learning method and heuristic is useful in practice, i.e. is able to make sensible suggestions,
2. determine to which extend additional information can be inferred when enriching ontologies with suggestions by the learning algorithm (described as *hidden inconsistencies* and *additional instances* below),
3. evaluate whether the method is sufficiently efficient to work on large real world ontologies,
4. assess the accuracy of the approximation method presented in Section 6.3.2 in practice.

We wrote an evaluation script, which takes an OWL file as input and works as follows: First, all classes with at least 3 inferred instances are determined. For each class A , the learning method generates at most ten suggestions with the best ones on top of the list. This is done for learning super classes ($A \sqsubseteq C$) and equivalent classes ($A \equiv C$) separately. If the accuracy of the best suggestion exceeds a threshold of 85%, we suggest them to the knowledge engineer. The knowledge engineer then has three options to choose from: 1. pick one of the suggestions by entering its number (accept), 2. declare that there is no sensible suggestion for A in her opinion (reject), or 3. declare that there is a sensible suggestion, but the algorithm failed to find it (fail). If the knowledge engineer decides to pick a suggestion, we query whether adding it leads to an inconsistent ontology. The consistency test is performed using a standard OWL reasoner – Pellet in our case. We call this case the discovery of a *hidden inconsistency*, since it was present before, but can now be formally detected and treated. We also measure whether adding the suggestion increases the number of inferred instances of A . For instance, picking one of the suggestions in Example 1.1 may allow a reasoner to infer additional capital cities, e.g. those cities which are related via `isCapitalOf` to a country, but have not been explicitly asserted to the class `Capital`. Being able to infer *additional instances* of A therefore provides added value (see also the notion of *induction rate* as defined in [Fanizzi et al., 2008] for further reading).

We used the default settings of 5% noise and an execution time of 10 seconds for the algorithm. The knowledge engineers were two researchers, who made themselves familiar with the domain of the test ontologies. We are aware that an ideal

ontology	#logical axioms	#suggestion lists	accept (1) in %	reject (2) in %	fail (3) in %	selected position on suggestion list (incl. std. deviation)	avg. accuracy of selected suggestion in %	#hidden inconsistencies	#add. instances (equivalence only)	#add. instances total
SC ontology	20081	12	79	21	0	2.2±2.1	96.8	0	506.0	1771
Finance	16057	50	52	48	0	3.6±2.6	96.7	0	211.4	1162
Biopax L. 2	12381	34	78	22	0	2.7±2.2	99.5	1	229.4	803
Intergeo	8803	180	56	44	0	1.6±1.2	96.9	1	14.4	295
Economy	1625	22	74	26	0	1.5±0.9	93.2	0	12.8	77
Br. Cancer	884	77	56	44	0	3.7±2.6	96.7	1	7.1	82
Eukariotic	38	8	91	9	0	2.5±1.2	92.1	0	2.4	7

Table 7.5: Evaluation results on several ontologies.

evaluation procedure would require OWL knowledge engineers from the respective domains, e.g. different areas within biology, medicine, finance, and geography. Considering the budget limitations, however, we believe that our method is sufficient to be able to meet the four evaluation objectives mentioned above. Each researcher worked independently and had to make 383 decisions (between select, reject, and fail) overall. The time required to make those decisions was 40 working hours per researcher. Both researchers obtained similar results. The evaluation machine was a notebook with a 2 GHz CPU and 3 GB RAM.

Table 7.5 shows the evaluation results. All ontologies were taken from the Protégé OWL and TONES repositories. Evaluation candidates in these repositories were those ontologies, which could be loaded and classified by Pellet in reasonable time and contain several classes with at least three instances. Between the candidate ontologies, we selected seven ontologies, which vary in size and complexity. For the sake of simplicity, we summed up values for generating equivalence and super class axioms, e.g. the third column is the sum of suggestion lists for equivalence axioms and super class axioms.

Objective 1: We can observe that the researchers picked option 1 (accept) most of the time, i.e. in many cases the algorithm provided meaningful suggestions. The researchers never declared that the algorithm failed on finding a potential solution. This allows us to positively answer the first evaluation objective. The 7th column shows that many selected expressions are amongst the top 5 (out of 10) in the suggestion list, i.e. providing 10 suggestions appears to be a reasonable choice.

Objective 2: In three cases a hidden inconsistency was detected. Both researchers independently coincided on those decisions. A low number of inconsistencies was expected due to the high quality of the knowledge bases. The last column shows that in all ontologies additional instances could be inferred for the classes to describe if the new axiom would be added to the ontology after the learning process. Overall, being able to infer additional instances was very common and hidden inconsistencies could sometimes be detected.

In the second part of our evaluation, we measured the impact of optimisation steps (see Section 6.3.2). To do this, we used a similar procedure as above, i.e. we processed the same ontologies and measured for how many class expressions we can measure a heuristic value within the execution time of 10 seconds. For each ontology, we averaged this over all classes with at least three instances. We did this for four different setups by enabling/disabling the stochastic heuristic measure and enabling/disabling the reasoner optimisations. We used Pellet 2.0 as underlying reasoner. The test machine had a 2.2 GHz dual core CPU and 4 GB RAM.

Objective 3: The results of our performance measurements are shown in Table 7.6. If evaluating a single class expression would take longer than a minute (the algorithm does not stop during such an evaluation), we did not add an entry to the table. We can observe that the approximate reasoner and the stochastic test procedure both lead to significant performance improvements. Since we prefer closed world reasoning as previously explained, the approximate reasoner would be a better choice even without improved performance. The performance gain of the stochastic methods is higher for larger ontologies. Apart from better performance on average, we also found that the time required to test a class expression shows smaller variations compared to the non-stochastic variant. Overall, a performance gain of several orders of magnitudes has been achieved. One can conclude that without approximate reasoning and stochastic coverage tests, the learning method would not work reasonably well on large ontologies.

Objective 4: To estimate the accuracy of the stochastic coverage tests, we evaluated each expression occurring in a suggestion list using the stochastic and non-stochastic approach. The last column in Table 7.6 shows the average absolute value of the difference between these two values. It shows that the approximation differs by less than 1% on average from an exact computation of the score with low standard deviations. This means that the results we obtain through the method described in Section 6.3.2 are very accurate and there is hardly any influence on the learning algorithm apart from improved performance.

Remark 7.1 (Extended Evaluation)

The first ontology engineering evaluation presented here will be extended in future work. The following additions will be made: 1.) Five different heuristics will be employed and tested. In addition to the one used here, F-measure, generalised F-measure [d’Amato et al., 2008a], Jaccard distance, and predictive accuracy will be evaluated in terms of user satisfaction. User satisfaction will be measured using a standard five-star rating system. 2.) The use of the approximate reasoner

ontology	#tests stochastic		#tests non-stochastic		stoch. diff. (\pm std. dev.)
	appr. reas.	stand. reas.	appr. reas.	stand. reas.	
SC Ontology	20 800	—	630	—	0.6% \pm 0.8%
Finance	71 400	72	29 000	—	0.1% \pm 0.3%
Biopax L. 2	20 700	35	2 200	—	0.2% \pm 0.6%
Intergeo	77 100	—	26 000	—	0.4% \pm 0.8%
Economy	120 000	8 100	40 000	—	0.5% \pm 0.5%
Br. Cancer	116 000	2 400	83 000	1 300	0.1% \pm 0.4%
Eukariotic	123 000	8 900	116 000	5 800	0.0% \pm 0.0%

Table 7.6: Performance assessment showing the number of class expressions, which are evaluated heuristically within 10 seconds, with stochastic tests enabled/disabled and approximate reasoning enabled/disabled. The last column shows how much heuristic values computed stochastically differ from real values. Values are rounded.

(see Section 6.3.1) compared to a standard reasoner will be evaluated analogously by user ratings. 3.) The evaluation will be performed by a higher number of evaluators and cover more ontologies.

Currently, the five heuristics have been implemented in the DL-Learner framework, a dedicated graphical user interface has been developed for the extended evaluation, and five researchers have agreed on serving as evaluators. The evaluation is scheduled for the first months in 2010. In a later extension of the evaluation, knowledge engineers may be invited to test the algorithm on a set of domain-specific ontologies. \square

7.4 Fragment Extraction Evaluation

In Section 6.2, we presented a technique for extracting a knowledge base fragment with respect to a learning problem in order to solve it more efficiently. The evaluation of this fragment extraction is split into two parts. In the first part, we evaluated the performance of the SPARQL retrieval component and the OCEL learning algorithm. The results are depicted in Figure 7.10. We randomly selected ten YAGO classes in DBpedia and retrieved instances that belong to the class as positive examples and then selected the same number of negative examples from a super class. We performed an extraction with varying recursion depth, which is the most important factor influencing performance, and recorded the following values: number of triples extracted (left figure), time needed for extraction (right figure, lower line of each color), and total time needed for extraction and learning (right figure, upper line of each color). Please note that a recursion depth of, e.g. two, includes all instances at distance smaller or equal two plus the complete class hierarchy spawned by these instances. The optional parameters *Get all*

superclasses and *Close after recursion* were enabled during the post-processing. Each point in the figure is an average over 10 runs and was obtained using a Virtuoso DBpedia mirror on our local network running on a 2.4 GHz dual core machine with 4 GB memory.

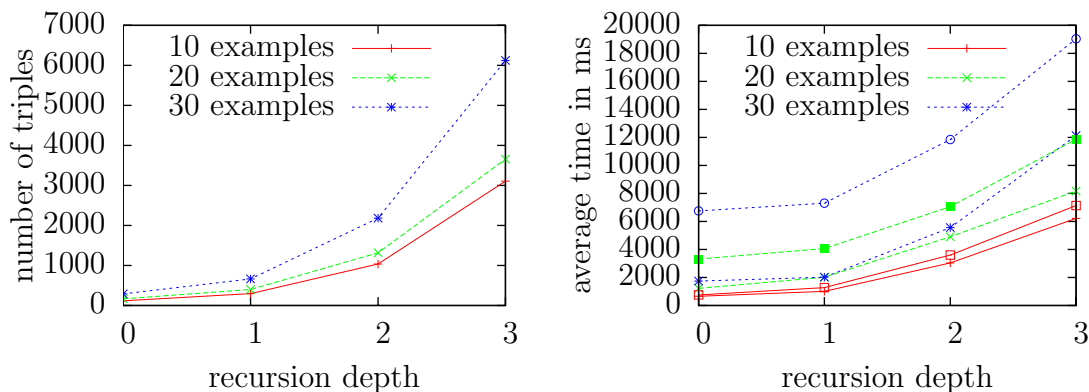


Figure 7.10: Left: extracted triples depending on recursion depth and number of examples. Right: extraction time needed depending on recursion depth and number of examples - for each color the lower line is the time needed for extraction while the upper line is the total time (including learning).

We can see that the curves for the time of extraction and learning in the right figure is equally or less steep than the increase in number of extracted triples in the left figure. The time for the learning process increases with more examples used, not only because of the increased time needed for reasoning but also due to the fact that the learned class expressions tend to get more complex for a higher number of examples. Overall, we achieved typical total learning times on a very large and dense (more than 8 properties associated to an instance on average) DBpedia knowledge base of a couple of seconds. Performance could be improved further by merging several SPARQL queries into more complex ones such that the triple store can make use of further internal optimization routines.

In the second part of our evaluation, we measured the validity of learned class expressions on the fragment, when compared to the whole ontology. As mentioned before, we choose the Semantic Bible ontology²¹ as target, because it is a medium sized ontology, contains complex background knowledge and is still manageable by a reasoner as a whole. It consists of 49 classes, 724 instances, 29 object properties and 9 data properties (4350 axioms total) and is in OWL-DL (but not OWL-Lite). The ontology also contains a variety of property axioms (domain: 35, range: 35, inverse: 17, symmetric: 6, subproperty: 12, functional: 4). To objectively compare the fragment selection approach with the normal approach we randomly selected 100 different sets of learning problems with 10 instances each (5 positive example

²¹<http://www.semanticbible.com/ntn/ntn-overview.html>

Semantic Bible	S_10s	N_10s	S_100s	N_100s
acc fragment(%)	67.8 (± 15.5)	61.4 (± 12.0)	73.7 (± 13.7)	67.4 (± 14.6)
acc whole (%)	67.6 (± 15.4)	61.4 (± 12.0)	73.5 (± 13.7)	67.4 (± 14.6)
acc pos (%)	100.0 (± 0)	100.0 (± 0)	100.0 (± 0)	100.0 (± 0)
acc neg (%)	35.2 (± 30.9)	22.8 (± 24.0)	47.0 (± 27.4)	34.8 (± 29.2)
extraction time	1.2s (± 0.4 s)	.0s (± 0 s)	1.3 (± 0.7)	.0s (± 0 s)
reasoner init time	.1s (± 0.1 s)	.2s (± 0 s)	.0 (± 0.1)	.3s (± 0 s)
learning time	10.5s (± 0.7 s)	27.1s (± 0.7 s)	102.3 (± 4.7)	107.0s (± 16.9 s)
axiom number	726 (± 221)	4350	726 (± 221)	4350
desc. length	3.8 (± 3.0)	2.2 (± 1.6)	5.5 (± 3.8)	3.6 (± 2.7)
desc. depth	2.1 (± 1.1)	1.6 (± 0.7)	2.8 (± 1.5)	2.2 (± 1.2)
Semantic Bible	S_1000	N_1000	S_10000	N_10000
acc fragment(%)	65.3 (± 14.7)	62.0 (± 13.0)	67.9 (± 15.5)	69.2 (± 15.0)
acc whole (%)	65.1 (± 14.6)	62.0 (± 13.0)	67.7 (± 15.4)	69.2 (± 15.0)
acc pos (%)	100.0 (± 0)	100.0 (± 0)	100.0 (± 0)	100.0 (± 0)
acc neg (%)	30.2 (± 29.2)	24.0 (± 26.1)	35.4 (± 30.9)	38.4 (± 30.0)
extraction time	1.1s (± 0.3 s)	.0s (± 0 s)	1.2s (± 0.4 s)	.0s (± 0 s)
reasoner init time	.0s (± 0 s)	.3s (± 0.2 s)	.1s (± 0 s)	.3s (± 0 s)
learning time	3.7s (± 2.3 s)	52.6s (± 56.9 s)	27.9s (± 14.1 s)	292.8s (± 92.7 s)
axiom number	726 (± 221)	4350	726 (± 221)	4350
desc. length	3.2 (± 2.5)	2.4 (± 1.8)	3.6 (± 2.9)	4.1 (± 2.8)
desc. depth	1.9 (± 1.0)	1.6 (± 0.8)	2.2 (± 1.2)	2.3 (± 1.2)

Table 7.7: The table shows the statistics for the fragment selection (S) approach compared to the “normal”(N) usage of the learning algorithm. We tested fixed runtime (10 seconds and 100 seconds) and fixed number of concept tests (1000 and 10000). All values are averaged over the same 100 example sets. Standard deviation is given in brackets. A 2.4 GHz dual core machine with 4 GB memory was used and the fragment was retrieved via SPARQL from a local Joseki endpoint.

instances and 5 negative example instances)²² and conducted the experiments with the same learning algorithm configuration and the same underlying reasoner (Pellet). In the first 4 experiments (S_10s, N_10s, S_100s, N_100s) the learning algorithm was stopped after a fixed time period (10 seconds and 100 seconds) and the best learned concept so far was validated versus the whole ontology. In the remaining 4 experiments (S_1000, N_1000, S_10000, N_10000) the algorithm was stopped after a fixed number of concept tests (cf. Figure 2.4, generate and

²²Random selection is different from real life problems. However, it is sufficient to gain some insights w.r.t. scalability.

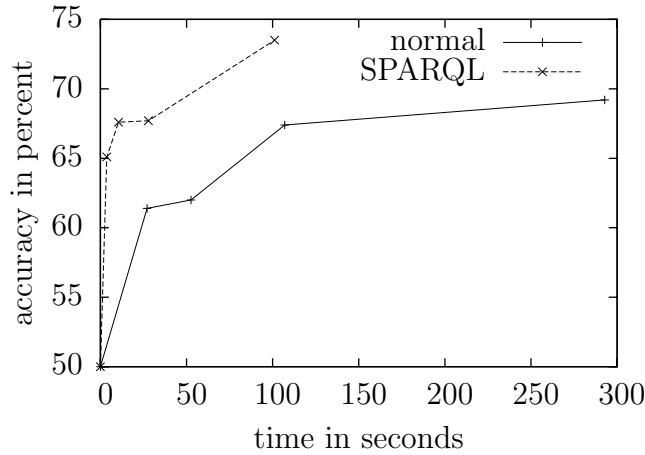


Figure 7.11: Time vs. Accuracy for learning on the Semantic Bible ontology. The two lines are for using only a fragment of the ontology or using the complete ontology.

test approach) independently of time needed. The fragment was extracted with the following parameters: recursion depth 2, close after recursion enabled, get all superclasses enabled, get explicit property information, no filters, literals allowed. The result can be viewed in Table 7.7.

The setup of the experiment is meant to answer two questions. First, we wanted to know how large the actual error is, if the fragmentized approach returns a learned class and analyze if we correctly predicted the type of error that can occur and second, we wanted to compare performance (fragment vs. whole).

Because the learning algorithm uses top-down refinement and ignores all class expressions that do not cover all positive examples if the noise parameter is set to 0%, the accuracy on the positive examples only is always stable at 100%. This is also true for the fragment because of monotonicity of description logics. The small error of 0.2% occurred, as predicted, when previously not covered negatives were covered in the whole ontology. We manually checked the data and found that a part of the learned class expression ($Object \sqcup \exists locationOf.\top$) contained an inverse functional property with only an inbound edge to the example instance, which is not covered on purpose by our extraction method (cf. point 6 "Inferred Property Information" on page 123).

The low overall accuracy of the class expressions (only 60% to 75%) is due to the schematic similarity between random sampled individuals, which made it impossible to induce sensible class expressions. The benefit in runtime of the fragmentized approach can be seen in Figure 7.11. We would like to note again that we choose the medium-sized Semantic Bible ontology for evaluation. The real target of the fragment selection approach are even larger knowledge bases, which currently only support minimal reasoning mechanisms, if any. The experiments showed an improvement in runtime by roughly the factor 10 without losing quality.

The highest accuracy (73.7%) by a small margin in the set time frame was achieved by the reasoning over the fragment. The average description depth (2.8) and length (5.5) also reveals that it is possible to construct complex class expressions with the information contained in the fragment.

7.5 Further Applications

Apart from the described applications (ontology engineering and carcinogenesis), there are further projects, which are work in progress at the time the thesis was finished. All of them are in an advanced state and therefore briefly described here.

7.5.1 Predictions of the Effect of Mutations on the Protein Function

Understanding how genetic alterations, also called mutations, affect genes at a molecular level is a challenge in biology and medicine. The SM2PH-db²³ ("from Structural Mutation to Pathology Phenotypes in Human-database") is a database for investigating the structural and functional impact of missense variants with respect to their phenotypic effects in the context of human monogenic diseases. Similarly to the carcinogenesis task, we converted this knowledge to OWL, which resulted in a knowledge base with 21 classes, 17 properties, 9 435 individuals, and more than 200 000 axioms. The automatic conversion script allows to keep in sync with the SM2PH database, which is updated bimonthly. The work was carried out in cooperation with the University of Strasbourg.

Based on this ontology, a learning problem was posed, where the positive examples are those mutations leading to diseases and the negative examples are those mutations not leading to diseases. The learned results provide insights as to what properties are likely to lead to diseases.

A particular challenge in this case is that there are thousands of examples. The approximate coverage test procedure described in Section 6.3 turned out to provide order of magnitude improvements for this problem. Specifically, the runtime is about 1% compared to not approximating coverage. Most other ILP tools do not work with such a high number of examples [Watanabe and Muggleton, 2009]. Similar as in the carcinogenesis case, DL-Learner usually derives shorter and more readable hypothesis. The exact assessment of the performance of DL-Learner methods compared to other machine learning tools and specific algorithms developed for solving this problem is the subject of currently ongoing work. Early results indicate that the problem is very challenging, but OCEL is apparently competitive with the state of the art in this area.

²³<http://decryphon.igbmc.fr/sm2ph>

7.5.2 NLP2RDF

NLP2RDF²⁴ is a framework for integrating existing natural language processing (NLP) approaches into a data-driven architecture. It implements an NLP pipeline incorporating various methods ranging from low-level morphology analysis to high-level anaphora resolution. The output of each step is aggregated in RDF and structured with existing linguistic ontologies. In addition, the generated RDF is enriched with background knowledge from the Web of Data, thus mixing syntactical explicit information with semantic information. As a prerequisite, statistical data from the Wortschatz project²⁵ was converted to RDF and mapped to DBpedia.

DL-Learner algorithms are used in this project to classify text, e.g. blog entries, based on the generated background knowledge via the NLP pipeline briefly described above. In linguistics, this can be used to identify typical properties of language structures. As an example, OCEL was used to learn that passive sentences often have the following structure (using the namespace `nlp2rdf` for <http://nlp2rdf.org/ontology/>):

$$\begin{aligned} &(\text{nlp2rdf:Sentence} \sqcap \exists \text{nlp2rdf:syntaxTreeHasPart}. \\ &\quad (\text{nlp2rdf:VVPP} \sqcap \exists \text{nlp2rdf:previousToken}. \\ &\quad \quad (\text{nlp2rdf:APPR} \sqcup \text{nlp2rdf:VAFIN}))) \end{aligned}$$

The classes VVPP, APPR and VAFIN were generated by a POS-tagger based on sample sentences from the Negra corpus²⁶ with the Stuttgart/Tübinger Tagset²⁷. According to definitions in a used linguistic ontology²⁸ VVPP's are "past participles", APPR are "temporal, causal, modal and local prepositions" and VAFIN "finite auxiliary verbs". Thus, the learned definition states that a passive sentence is a sentence, which contains an APPR or VAFIN followed by a VVPP.

The project was initiated by Sebastian Hellmann. The thesis author is currently a minor contributor in it. The (not yet proven) hypothesis of NLP2RDF is that aggregating the output of different NLP tools in a structured format and enriching it using the Web of Data allows to achieve better results than previous NLP approaches for many tasks.

7.5.3 ORE - Ontology Repair and Enrichment

ORE (ontology repair and enrichment) is a DL-Learner based tool for repairing and enriching OWL ontologies. It supports the detection of a range of ontology modelling problems and guides the user through the process of resolving them.

²⁴<http://code.google.com/p/nlp2rdf/>

²⁵<http://wortschatz.uni-leipzig.de>

²⁶<http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/negra-corpus.html>

²⁷<http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/stts.asc>

²⁸<http://141.89.100.105/owl2/stts.owl>

In the first phase, consistency of an input ontology is checked. If the ontology is not consistent, reasoner explanations [Horridge et al., 2008] are displayed. The user can then remove or edit axioms, which amounts to the creation of a repair plan. Similar methods are used for handling unsatisfiable classes in the second phase, which are often modelling errors as well. Ranking methods allow to detect problematic axioms and suggest actions to resolve them. In the third phase, the enrichment process, the CELOE learning algorithm loops through classes in the ontology. If promising suggestions, i.e. those exceeding a certain accuracy threshold, can be made, they are presented to the user. If the user accepts a suggestion, then the tool checks for consequences of adding the axiom. Since, according to the study in Section 7.3, in many cases the selected expressions do not have 100% accuracy, it is interesting to look at not covered instances of the class as well as looking at covered entities not belong to the class. The ORE tool displays those cases and suggests possible changes depending on the context. It can be useful to delete an entity, "move" it "up" or "down" the class hierarchy, provide more information about an entity etc. The result of applying ORE to an ontology is a consistent and expressive knowledge base. It can work with very large knowledge bases using the SPARQL fragment extraction method presented in Section 6.2 as well. In this case, a relevant OWL class, which should be investigated, has to be specified such that the corresponding knowledge base fragment can be extracted. More information can be found at <http://dl-learner.org/wiki/ORE>.

7.5.4 moosique.net - Music Recommendations

The described learning algorithms can also be used for providing recommendations. Given a set of interesting entities, they return an OWL class expression describing those entities (positive only learning). This idea was applied to the music recommendation use case. The background knowledge is the RDF version of the freely available Jamendo music database²⁹. On top of this relatively flat knowledge base, a schema covering various topics such as music style (rock, pop etc.) including their specific variants, instruments, producers, and other information was created. The tags assigned by Jamendo users to music albums were then converted into this schema. This process happens on the fly using the extraction technique presented in Section 6.2. It allows the application of learning algorithms like ELTL and CELOE, which can suggest further songs or albums given the ones someone previously listened to. This functionality was combined into a free internet radio, which can be accessed as a prototype at <http://moosique.net>.

²⁹<http://www.jamendo.com>

7.6 Strengths and Limitations of the Described Approaches

This section summarizes the advantages and disadvantages of the presented algorithms. It is organised along the criteria of applicability, accuracy, readability, scalability, and usability. Apart from analysing strengths and limitations of the concrete approaches, we also want to draw a more general picture on why learning in DLs is a promising line of research.

Applicability The Semantic Web is rapidly growing³⁰ and contains knowledge from diverse areas such as science, music, people, books, reviews, places, politics, products, software, social networks, as well as upper and general ontologies. The underlying technologies, sometimes called *Semantic Technologies*, are currently starting to create substantial industrial impact in application scenarios on and off the web, including knowledge management, expert systems, web services, e-commerce, e-collaboration, etc. Data exchange and integration is central to these technologies, which thus hinge on the use of suitable knowledge representation formalisms. Consequently, it is important to adhere to established standards, foremost those established by the World Wide Web Consortium (W3C). Since 2004 (2009), the Web Ontology Language OWL (OWL 2), has been the W3C-recommended standard for Semantic Web knowledge representation and has been a key to the growth of the Semantic Web. Being able to apply inductive learning on this data and to use OWL/DLs themselves as results of a learning algorithm widens the possibilities for ILP research and practice since it opens up the Semantic Web as field of application.

It could be argued that ILP systems based on logic programs can be used to achieve this task. However, OWL ontologies and logic programs are incomparable with respect to their expressiveness, i.e. there are OWL ontologies not expressible in Horn logic and vice versa. This means that the algorithms cannot be applied to all scenarios where Horn logic ILP programs can be used, which is both a strength and a limitation of the described approaches. One restriction are predicates with arity greater two. Since concepts in description logics correspond to predicates of arity one and roles correspond to predicates of arity two, it is not straightforward to express predicates with higher arity. However, in many cases, e.g. the benchmarks earlier in this chapter, this can still be done but usually requires a human expert. Even in such cases, it often turned out that knowledge can be represented in a human friendlier and more readable way in OWL/description logics.

It should be noted that in principle each learning problem in DLs can be solved by ILP systems based on expressive formalisms like first order logic, since DLs are a fragment of first order logic. However, it can, of course, be inefficient to do this due to the larger search space and higher complexity of reasoning or undecidability.

³⁰For instance, the semantic index Sindice (<http://sindice.com/>) grows steadily and now lists more than 10 billion entities from more than 100 million web pages.

[Badea and Nienhuys-Cheng, 2000] has discussed why ILP systems may not be appropriate to learn DL concepts even in those cases where they can be used in principle. The paper analyses this for the case of using prenex conjunctive normal form (PCNF) and states that the main problems are that 1.) a conversion to PCNF can lead to an exponential blowup in knowledge base size and 2.) many formulae in PCNF do not have a counterpart in description logics, i.e. they are too fine grained. Similar arguments apply when using Horn logic or first order logic. On the other hand, using systems for less expressive formalisms cannot make use of all the carefully selected features of OWL.

OWL and description logics also enable *new application tasks*. One such task is ontology engineering, in particular suggesting definitions and super class axioms in knowledge bases based on instance data. We pursued this line of research by developing plugins for the popular Protégé and OntoWiki ontology editors. There have been heated discussions in the past on the use of logic programming versus description logics in ontology engineering, see e.g. [de Bruijn et al., 2005, Patel-Schneider and Horrocks, 2007] for recent developments. Also relevant are efforts for combining DLs and rules [Krötzsch et al., 2008a, Krötzsch et al., 2008b]. However, the growing popularity of the Web Ontology Language OWL seems to indicate that some of their distinctive features make it a viable and perhaps even superior alternative to logic programs in many application domains. This can in part be attributed to the fact that DLs restrict the modeller more severely in his use of the modelling language: Horn logic is Turing complete [Šebelík and Štěpánek, 1982] (and can, e.g. in the form of Prolog, even be used as a programming language), while DLs are usually decidable.

Accuracy We have shown to be more accurate than other DL learning systems and we claim to be more accurate than other general ILP tools for some learning problems. We have shown this for carcinogenesis, where DL-Learner was able to achieve statistically significantly higher accuracy than some state-of-the-art ILP systems. It is well-known that the choice of the target language is a critical one with respect to the accuracy and efficiency of learning algorithms. Hence, it is natural that DL concepts are more appropriate than other formalisms for a subset of learning problems. This particularly applies to domains, where ontologies are already widely used, e.g. the life sciences [Rector and Brandt, 2008, Belleau et al., 2008].

Readability As mentioned previously, the DL-Learner algorithms have a bias towards learning compact, readable concepts. We have shown that there are often orders of magnitude difference with respect to the size of the offered solutions in the carcinogenesis problem. Hence, we do consider readability to be a strength of our approach. Furthermore, DLs lend themselves easily to conversion between (controlled) natural language and formal language, see [Schwitter et al., 2008, Völker et al., 2007a] for references. This helps to close the gap between ontology engineers and domain experts.

Scalability Our experiments indicate that the presented approach can work efficiently with knowledge bases containing around 100.000 axioms – of course depending on the complexity of those axioms. This is sufficient for many realistic application scenarios. Furthermore, we also extended DL-Learner by a component, which allows to apply it to very large knowledge bases by selecting relevant knowledge fragments in a pre-processing step. We applied the procedure to DBpedia (see Section 6.1), containing more than 400 million axioms, and other large knowledge bases.

Usability DL-Learner requires only a minimal set of parameters to work well. Those parameters are the used background knowledge bases (which can be more than one), the positive and negative examples, and a termination criterion (e.g. minimum accuracy through the noise parameter). As the approaches are, in the case of OCEL, ELTL Base, and CELOE, anytime algorithms, bounding their maximum runtime can be convenient in systems which need to process several learning problems reliably without the risk of using too many resources. Also note that other tools like Aleph often require so called mode declarations in order to work efficiently by restricting the search space. These restrictions are already present in DL/OWL knowledge bases through domain and ranges of roles. They are used by the refinement operator automatically, which makes it easier to apply DL-Learner without the need to specify further restrictions (which require knowledge about the domain at hand). In fact, the DL-Learner ontology editor plugins can be invoked using a single mouse click and provide sensible suggestions.

Summary In summary, we argue that learning in DLs is limited in that not all typical ILP problems can be solved. However, it is also apparent that it can widen the scope of ILP to new application areas and tasks, in particular in the context of Semantic Web applications, which hinge critically on the employed knowledge representation formalisms. We have shown that DL-Learner is competitive with respect to accuracy and scalability with state-of-the-art ILP systems. We also claim that the provided solutions are more readable and that DL-Learner is easy to use.

8 Related Work

While related work has been mentioned throughout the thesis, this chapter serves as a central collection of material, which is related to and/or has influenced the work. It is organised in several categories corresponding to the closest research areas.

8.1 Inductive Learning in Description Logics

Research in this area started in the early 90s, but only recently gained momentum due to the rise of the Semantic Web. As one of the closely related works, in [Badea and Nienhuys-Cheng, 2000] a refinement operator for $\mathcal{AL}\mathcal{ER}$ has been designed to obtain a top-down learning algorithm for this language. Properties of refinement operators in this language were discussed and some claims were made, but a full formal analysis was not performed. The article also investigates some theoretical properties of refinement operators. As we have done with the design of ρ , they favour the use of a downward (as opposed to upward) refinement operator to enable a top-down search. The authors use $\mathcal{AL}\mathcal{ER}$ *normal form*, which is easier to handle than e.g. \mathcal{ALC} negation normal form, because $\mathcal{AL}\mathcal{ER}$ is not closed under boolean operations. As a consequence, they obtain a simpler refinement operator for which it is not clear how it could be extended to more expressive DLs. Our operator ρ , in contrast, lends itself much more easily to such extensions. We also deal quite differently with infinity, we show how the subsumption hierarchy of atomic concepts and roles can be used, use domain and range of roles to structure the search, and we describe how redundancy can be avoided efficiently. Moreover, our theoretical results are more general, i.e. covering more description languages and property combinations. In contrast to [Badea and Nienhuys-Cheng, 2000], we provided proofs, which were not available on request by the authors, and refuted one of their results. As mentioned before, the algorithm in [Badea and Nienhuys-Cheng, 2000] was not implemented, which is why we could not assess its performance on learning examples in our evaluation.

In [Esposito et al., 2004, Iannone and Palmisano, 2005, Iannone et al., 2007], algorithms for learning in description logics, in particular for the language \mathcal{ALC} , were created, which also make use of refinement operators – however, not as centrally as in our approach. The core idea of those algorithms is blame assignment, i.e. to find and remove those parts of a concept responsible for classification errors. In particular, [Iannone et al., 2007] described how to apply the

learning problem for classifying scientific papers. Instead of using the classical approach of combining refinement operators with a search heuristic, a different approach is taken therein for solving the learning problem by using approximated MSCs (most specific concepts). The most specific concept of an individual is the most specific class expression, such that the individual is instance of the expression. Empirically, a problem of these algorithms is that they tend to produce unnecessarily long concepts. One reason is that MSCs for \mathcal{ALC} and more expressive languages do not exist and hence can only be approximated. Previous work [Cohen et al., 1993, Cohen and Hirsh, 1994, Kietz and Morik, 1994] in learning in DLs has mostly focused on approaches using least common subsumers, which face this problem to an even larger extent according to their evaluation. The algorithms implemented in DL-Learner overcome this problem and investigate the learning problem and the use of top down refinement in detail. In our approaches, we also cannot guarantee that we obtain the shortest possible solution of a learning problem. However, the learning algorithms were carefully designed to produce short and readable solutions. For OCEL, the produced solutions will be close to the shortest solution in negation normal form.

DL-FOIL [Fanizzi et al., 2008] is a similar approach, which is based on a mixture of upward and downward refinement of class expressions. They use alternative measures in their evaluation, which emphasize the difference between deductive and inductive reasoning and take the open world semantics of description logics into account. As a consequence, three cases can be distinguished for instance checks: An individual is instance of a concept (response +1), an individual is instance of the negation of a concept (response -1) or none of both can be inferred (response 0). This leads to the use of alternative measures for description logics, e.g. in [d’Amato et al., 2006, d’Amato et al., 2008b]. In instance checks, a *match* stands for the case when deductive and inductive classifier coincide. An *omission* stands for the case when the inductive method cannot determine concept membership (response 0), but the deductive classifier can infer membership (response ± 1). *Commission* stands for the case when the deductive and inductive classifier disagree (+1 vs. -1 or -1 vs +1). The *induction rate* stands for those cases, where the inductive classifier determines membership (response ± 1), but this is not deductively derivable from the knowledge base (response 0). The latter is similar to the “additional instances” columns in Table 7.5. As mentioned in Section 2.2.2, DL-Learner allows to switch between a three valued learning problem and the two valued problems we used throughout the thesis. One reason why we used the two valued problem was that this allows to compare the results to other ILP tools. However, it is worthwhile to use the proposed alternative measures and their investigation is an interesting area of future work. In particular, [d’Amato et al., 2008a] recently introduced generalised F-measure, which can be used as a score function in the learning algorithms. Those measures are currently integrated in DL-Learner and will later be evaluated.

[Esposito et al., 2004] and [Fanizzi et al., 2004] stated that an investigation of the properties of refinement operators in description logics, as we have done in this

thesis, is required for building a theoretical foundation of the research area. In [Fanizzi et al., 2004] downward refinement for \mathcal{ALN} was analysed using a clausal representation of description logic concepts. Refinement operators have also been dealt with within hybrid systems. In [Lisi and Malerba, 2003] ideal refinement for learning \mathcal{AL} -log, a language that merges DATALOG and \mathcal{ALC} , was investigated. Based on the notion of \mathcal{B} -subsumption, an ideal refinement operator was created. This line of research was pursued further in [Lisi, 2008] and in [Lisi and Esposito, 2008] applied in the context of ontology evaluation.

8.2 Refinement Operators

In this section, refinement operators outside of description logics are discussed. In the area of Inductive Logic Programming considerable efforts have been made to analyse the properties of refinement operators (for a comprehensive treatment, see e.g. [Nienhuys-Cheng and de Wolf, 1997]). In general, applying refinement operators for clauses to solve the learning problem in DLs is usually not a good choice (see [Badea and Nienhuys-Cheng, 2000] and Section 7.6). However, the theoretical foundations of refinement operators in Horn logics also apply to description logics, which is why we want to mention work in this area here.

A milestone in Machine Learning [Mitchell, 1997] in general was the Model Inference System in [Shapiro, 1991]. Shapiro describes how refinement operators can be used to adapt a hypothesis to a sequence of examples. Afterwards, refinement operators became widely used as a learning method. The properties in Definition 2.29 were defined as theoretical criteria for the quality of refinement operators. In [van der Laag and Nienhuys-Cheng, 1994] some general results regarding refinement operators in quasi-ordered spaces were published. Nonexistence conditions for ideal refinement operators relating to infinite ascending and descending refinement chains and covers have been developed. This has been used to show that ideal refinement operators for clauses ordered by θ -subsumption do not exist. Unfortunately, we could not make use of these results directly, because proving properties of covers in description logics without using a specific language is likely to be harder than directly proving the results.

[Nienhuys-Cheng et al., 1993] discussed refinement for different versions of subsumption, in particular weakenings of logical implication. A few years later, it was shown in [Nienhuys-Cheng et al., 1999] how to extend refinement operators to learn general prenex conjunctive normal form. Perfect operators, i.e. operators which are weakly complete, locally finite, non-redundant, and minimal, were discussed in [Badea and Stanciu, 1999]. Since such operators do not exist for clauses ordered by θ -subsumption [van der Laag and Nienhuys-Cheng, 1994], weaker versions of subsumption were considered. This was later extended to theories, i.e. sets of clauses [Fanizzi et al., 2003]. A less widely used property of refinement operators, called flexibility, was discussed in [Badea, 2000]. Flexibility essentially means that previous refinements of an operator can influence the choice of the

next refinement. The article discusses how flexibility interacts with other properties and how it influences the search process in a learning algorithm.

8.3 (Semi-)Automatic Ontology Engineering

Regarding (semi-)automatic ontology engineering, the line of work starting in [Rudolph, 2004] and further pursued e.g. in [Baader et al., 2007b] investigates the use of formal concept analysis for completing knowledge bases. It is promising, but targeted towards less expressive description logics and may not be able to handle noise as well as a machine learning technique. [Völker and Rudolph, 2008] proposes to improve knowledge bases through relational exploration and implemented it in the RELExO framework¹. It is complementary to the work presented here, since it is focused on simple relationships and the knowledge engineer is asked a series of questions. The knowledge engineer either has to positively answer the question or provide a counterexample. A different approach to learning the definition of a named class is to compute the MSCs for all instances of the class. One can then compute the least common subsumer (lcs) [Baader et al., 2007c] of those expressions to obtain a description of the named class. However, in expressive description logics, an msc need not exist and the lcs is simply the disjunction of all expressions. For light-weight logics, such as \mathcal{EL} , the approach appears to be promising. [Völker et al., 2007b] focuses on learning disjointness between classes in an ontology to allow for more powerful reasoning and consistency checking. In [d’Amato et al., 2008b], inductive methods have been used to answer queries and populate ontologies using similarity measures and a k-nearest neighbour algorithm. Along this line of research, [d’Amato et al., 2005] defines similarity measures between concepts and individuals in description logic knowledge bases.

Naturally, there is also a lot of research work on ontology learning from text. The most closely related approach in this area is [Völker et al., 2007a], in which OWL DL axioms are obtained by analysing sentences, which have definitorial character.

Another interesting area of related research are natural language interfaces. In [Cimiano et al., 2009], so called intensional answers are investigated. For instance, a query “Which states have a capital?” can return the name of all states as intensional answer or “All states (have a capital).” as extensional answer. Similarly, the query “Which states does the Spree flow through?” could be answered by “All the states which the Havel flows through.”. The intensional answers of such queries can sometimes reveal interesting knowledge and they can also be used to detect flaws in the knowledge base. The authors argue that this form of query based ontology engineering can be useful. This line of research is closely related to this PhD thesis. ILP techniques (specifically LGGs similar to the Golem system briefly described in Section 2.2.1) are used to solve it algorithmically. The target language is F-Logic and a bottom up approach is used in contrast to the

¹<http://relexo.ontoware.org/>

top-down refinement driven techniques presented in our work. An application of DL-Learner algorithms to intensional query answering in the Semantic Web would be straightforward and an interesting area of future work.

8.4 Knowledge Fragment Selection

Closely related to the presented fragment extraction are ABox contraction techniques. [Fokoue et al., 2006], for example, present an approach how to compute a possibly much smaller summary of an ABox obeying equivalent reasoning properties. Such approaches are suitable for clean and homogeneous ontologies with small TBoxes and large ABoxes, while our approach is targeted at impure, heterogeneous multi-domain ontologies with both components, TBoxes and ABoxes, being large. Another application, that is concerned with reasoning on large ABoxes is instanceStore [Horrocks et al., 2004], which as of now only works on role-free knowledge bases. A project aiming to enable massive distributed incomplete reasoning is LarKC [Fensel et al., 2008], which is currently in progress.

A further related approach is described in [Seidenberg and Rector, 2006], where fragments of the GALEN ontology are extracted to enable efficient reasoning. The major difference compared to our approach is that we focused on providing a fragment extraction algorithm suitable for learning class expressions. We start from instances instead of classes and do not need to extract subclasses of obtained classes. Our approach was implemented with support for SPARQL and Linked Data for querying knowledge bases. Furthermore, we do not require the OWL ontology to be normalized and can handle complex class expressions as fillers of property restrictions. Similarities between both approaches are the idea of a (recursion) depth limit and the extraction of class and property hierarchies.

9 Conclusions and Future Work

In this chapter, we summarize the research work and highlight the main results. We organise the chapter according to the overall thesis structure. First, we discuss refinement operators and, secondly, the learning algorithms incorporating those operators with an emphasis on scalable extensions. Finally, we conclude on evaluation and use cases. This is followed by an outlook on future work in the research area.

9.1 Refinement Operators

We performed an extensive analysis of the properties of refinement operators in description logics. While such an analysis is difficult to accomplish without considering a specific description logic, we took care that our results are as general as possible, i.e. cover many description logics. We first argued why minimality, which is an important refinement operator property in logic programs, is unlikely to play a key role in description logics. We showed that it is incompatible with weak completeness even for the very restricted description logic \mathcal{AL} . We proved several positive and negative results concerning completeness, properness, redundancy, and finiteness of refinement operators leading to Theorem 3.16. We believe this theorem to be a significant theoretical result that serves as a starting point for practical considerations when using refinement operators in DLs. An important insight is that there are no ideal operators for many expressive description logics, which indicates that learning in expressive DLs is challenging.

In general, combining completeness of an operator with one of the other (positive) properties turned out to be an intricate problem in practice. A positive result could, however, be obtained for the light-weight description logic \mathcal{EL} . Its beneficial reasoning properties, i.e. polynomial time inference, could be carried over to the refinement task. In Section 4.2, we defined an ideal \mathcal{EL} refinement operator. We also showed that this operator is efficient even on large knowledge bases. Before the investigation of \mathcal{EL} refinement, we introduced a refinement operator with a very expressive target language. The completeness of this operator was shown and it was later extended with support for concrete roles and cardinality restrictions. We also pointed out how this operator can be turned into a proper and complete refinement operator.

9.2 Learning Algorithms and Scalability

A learning algorithm can essentially be designed by combining a refinement operator with an appropriate heuristic. When designing such algorithms, one of the lessons learned is that while theoretically the refinement operator can be treated as a black box, it is beneficial or sometimes even necessary to take its properties into account. For instance, the OCEL algorithm (Algorithm 3 on page 93) is designed to be able to handle infinite refinement operators by allowing a length-limited, stepwise refinement of nodes in the search tree. It also uses a mechanism to efficiently avoid redundancy by using a normal form and defining an ordering over concepts. OCEL is highly configurable and targets most of the standard learning problems.

Contrary, ELTL (Algorithms 4 and 5 on page 95ff) is much simpler in this respect, because it uses an ideal refinement operator. Since \mathcal{EL} does not contain disjunction, which is required for solving some learning problems, we optionally allow a covering approach to be used. Covering means to construct a disjunctive concept stepwise, which is inspired by ILP approaches. We also introduced a method which allows to simplify a solution by using a standard OWL reasoner in order to increase readability. ELTL mainly aims at simple learning scenarios. In particular, it is also targeted at the OWL 2 EL profile.

CELOE is an adapted version of OCEL and uses the same underlying refinement operator, but a different heuristic, which is designed for the ontology engineering use case. We motivate the heuristic by comparing it with a straightforward adaptation of learning from examples. Later, in Section 6.3.2, we showed how it can be computed efficiently in the presence of many examples.

After describing the learning algorithms, we tackled the question of scalability. We analysed that there are several reasons why learning algorithms may be slow. The first one is the sheer size of the hypothesis space. We devoted a large fraction of the thesis to refinement operators which can traverse this space in an ordered way by making use of the subsumption hierarchy. It became clear that employing as much background knowledge as possible is a key point for designing an efficient learning algorithm, since the knowledge can be used for structuring the search space and ruling out unsatisfiable hypothesis without needing to test them explicitly.

Apart from the search space problem, the part of a learning algorithm, which requires most processing time is the coverage test. There are two reasons for a coverage test to be slow: The first one is that OWL/DL reasoning is expensive. In particular, one hurdle is already to load a large knowledge base into a reasoner, which turns out to be impossible for many large knowledge bases. Therefore, we provided a fragment extraction algorithm, so that learning on the fragment is similar to learning on the whole knowledge base. This algorithm also tackles the problem that we cannot expect very large knowledge bases to be consistent. We lowered this barrier to requiring only that the considered fragment is consistent. These smaller fragments can be processed efficiently by an OWL reasoner.

We continued by arguing that a closed world assumption can be beneficial when performing coverage tests. We, therefore, designed an instance check procedure, which builds on a standard OWL reasoner, but uses a very fast, partial closed world evaluation procedure for instance checks in coverage tests. The second reason why coverage tests can be slow, is the potential presence of a high number of examples. We employed a stochastic approach for this case, which executes only as many instance checks as required for a reasonably good approximation of the score function. We showed that this approach can lead to an order of magnitude improvement in performance, while still being sufficiently accurate (see Table 7.6).

9.3 Implementation, Evaluation and Use Cases

A major outcome of this thesis is the DL-Learner software framework. By providing a toolset for learning OWL class expressions, we did not only establish the necessary basis for evaluating the presented algorithms. We also made the implementation transparent and developed a number of user interfaces to access their functionality. A number of students have worked on projects, which use DL-Learner: the OntoWiki and Protégé plugins (Section 7.3), the ORE tool, moosique.net, NLP2RDF (all in Section 7.5), and the DBpedia Navigator (Section 6.1.4).

After describing the implementation part in form of the DL-Learner project, we presented an evaluation involving several learning problems. In a first phase, we compared the algorithms against YinYang and genetic programming approaches, where it turned out that OCEL is clearly superior. ELTL was promising in those cases where its target language is sufficiently expressive. Afterwards we compared OCEL against state-of-the-art algorithms on a challenging learning problem. Its accuracy was the highest in this test and statistically significantly better than most approaches. However, the main advantage appears to be a higher readability of the learned hypothesis. We also evaluated the scalability improvements described in Chapter 6 and showed that order of magnitude improvements in performance are achieved. During our experiments, we experienced technical and engineering hurdles such as non-standard behaviour, lack of interlinking and semantic richness or simply inaccessibility. Hence, working with very large knowledge bases is still challenging from both - engineering and research - perspective.

Apart from the ILP problems, a major use case is ontology engineering. We presented plugins for OntoWiki and Protégé, which are seamlessly integrated and easy to use. The preliminary evaluation showed, with the mentioned limitations, that learning expressions in ontologies is a relevant problem and CELOE gave appropriate suggestions, which sometimes lead to the detection of modelling problems.

We concluded the evaluation in Chapter 7 with an overview of the strengths and limitations of the approaches with respect to the criteria applicability, accuracy, readability, scalability, and usability. In essence, we argue that our methods open up new application areas for Inductive Logic Programming theory and methods.

We also provide evidence that usability and readability are strengths. Regarding accuracy and scalability, we believe to be at least competitive with state-of-the-art machine learning tools.

9.4 Future Work

In the short- and mid-term future, one of the goals is to test the learning algorithms on a variety of scenarios in the spirit of the carcinogenesis problem in Section 7.2.2. We have already shown that this concrete learning problem can be solved at least equally well using the DL-Learner framework and description logics as knowledge representation formalism compared to using the state of the art in machine learning. An interesting research question is whether learning in description logics and OWL is also a viable alternative to ILP approaches in other cases. Predicting the effect of mutations (Section 7.5) is a specific problem, which we currently work on by using DL-Learner as well as other ILP approaches.

While we have focused on top down approaches in this thesis, this focus does not mean we consider bottom up approaches to be less interesting or relevant. Bottom up approaches usually have the advantage that they can learn complex hypotheses from few examples, since they directly employ the structure of examples as a starting point for the learning algorithm. In particular in the presence of limited computation resources, which do not allow testing the coverage of several thousand hypothesis, bottom up approaches can be a viable alternative. It might be particularly interesting to view examples as graphs, where a graph represents the neighbourhood of the example up to a certain depth. Using those graphs as starting points in a bottom up approach and utilizing graph operators for upward refinement could be a key to creating a different class of OWL/DL learning algorithms. This line of research could be promising, especially for positive only learning.

Another area for future extensions is the incorporation of fuzziness and probabilities. Fuzzy description logics, see e.g. [Straccia, 2001], allow to express vague concepts, whereas probabilistic description logics, see e.g. [Lukasiewicz, 2008] allow to specify the likelihood of an axiom being true. Both extensions are exciting in the context of concept learning. The recent success of probabilistic ILP [Raedt et al., 2008] is an additional motivation for pursuing further research in this direction.

In the ontology engineering use case, we have already laid theoretical and practical foundations by developing CELOE and deploying it in two plugins. The evaluation of CELOE revealed interesting and, in our opinion, promising results. However, a larger study would still be interesting. Particularly, we plan a more detailed evaluation of heuristics, e.g. generalised F-measure [d’Amato et al., 2008a], and the incorporation of domain experts. In a wider context, the approaches towards automatic and semi-automatic ontology engineering have been extended significantly in the past. Whereas ontology learning was often seen as automatic

learning from text, there are now several other methods available, which often require an interaction with the knowledge engineer. An important goal should be to turn those approaches into practice and disseminate them in the Semantic Web community. For this reason, the author of the present thesis plans to (co-)edit a book about (semi-)automatic ontology engineering with contributions from several researchers in this field.

Another potential area of future work is the use of learning approaches in recommender systems. Those systems usually employ user or item similarity for computing whether items preferred by one person are likely to be preferred by another person. Some recommender systems already make use of item descriptions, e.g. classification hierarchies in recommender systems [Ziegler et al., 2008]. However, the incorporation of learning algorithms would allow arbitrary OWL-compliant background knowledge, while still allowing a user to see why an item was suggested. To do this, recommender algorithms need to be combined with efficient learning algorithms. In particular, \mathcal{EL} as a light-weight description logic could play an important role in this area.

The fragment extraction approach described in Section 6.2 can be considered in a wider perspective. The creation of background knowledge is a tedious and time-consuming process. Many applications might prefer to choose a more light-weight approach of reusing larger ontologies, but only consider relevant fragments of them. In this sense, it may be an interesting research challenge to employ fragment extraction for Semantic Web applications and light-weight ontology engineering.

In correspondence with the interdisciplinary character of this thesis, which involves the Machine Learning as well as the Semantic Web research areas, our long-term vision is twofold: On the one hand, we hope to widen the scope of Machine Learning research, in particular Inductive Logic Programming, by making use of the large amount of knowledge in the Semantic Web. On the other hand, we want to improve the Semantic Web itself by making it easier to create and maintain expressive knowledge bases. We hope that this thesis constitutes a valuable contribution towards achieving these goals.

A Software Release History

The following software releases were made during the thesis.

- DL-Learner Machine Learning Framework:
 - Build 2009-05-06: Protégé plugin, DL-Learner manual, EL refinement operator, stochastic coverage estimation, CELOE
 - Build 2008-10-13: OCEL algorithm, GUI interface, approximate reasoner, carcinogenesis benchmark
 - Build 2008-02-18: introduction of component based structure, SPARQL support, OWL API reasoner support, Web Service interface
 - Build 2007-08-31: initial open source release (a number of algorithms already implemented)
- Protégé DL-Learner Plugin
 - 0.5.2 - released 2009-05-29 - bug fix release
 - 0.5.1 - released 2009-05-12 - bug fix release
 - 0.5 - released 2009-04-25 - proper integration in extended Protégé plugin mechanism, use of CELOE, result visualisation
 - 0.1 - released 2008-12-22 - initial version
- OntoWiki DL-Learner Plugin
 - contained in OntoWiki subversion and installable via plugin mechanism
- DBpedia Extraction
 - DBpedia 3.3 - released 2009-07
 - DBpedia 3.2 - released 2008-11
 - DBpedia 3.1 - released 2008-08
 - DBpedia 3.0 - released 2008-02
 - DBpedia 2.0 - released 2007-09
 - DBpedia 1.0 - released 2007-03

B DL-Learner Manual

DL-Learner is the open source machine learning framework, which contains the implementation of the techniques presented in this thesis. This appendix contains the relevant fractions of the DL-Learner manual as of September 2009. Note that parts of the manual may overlap with existing material in the thesis. However, the description is focused on software users and developers. The software DL-Learner has been accepted at the Open Source Track of the Journal of Machine Learning Research [Lehmann, 2009].

B.1 What is DL-Learner?

DL-Learner is an open source framework for (supervised) machine learning in OWL and description logics (from instance data). We further detail what this means:

OWL stands for “Web Ontology Language”. In 2004, it became the W3C¹ standard ontology language². As such it is one of the fundamental building blocks in the Semantic Web and has been used in several scenarios on and off the web. OWL is based on *description logics* (DLs), which are a family of knowledge representation languages. We refer to [Baader et al., 2007a] for an introduction to description logics. Since OWL formally builds on description logics, we can apply DL-Learner to knowledge bases in OWL or a variety of description languages.

Machine Learning is a subfield of Artificial Intelligence, which focuses on detecting patterns, rules, models etc. in data. Often, this involves a training process on the input data. In *Supervised* learning, this data is labelled, i.e. we are given a number of input-output mappings. Those mappings are also called *examples*. If the output is binary, then we distinguish positive and negative examples. DL-Learner as a framework is not restricted to supervised learning, but all algorithms currently build into it, are supervised.

In the most common scenario we consider, we have a background knowledge base in OWL/DLs and additionally, we are given positive and negative examples. Each example is an individual in our knowledge base. The goal is to find an OWL *class expression*³ such that all/many of the positive examples are *instances* of this expression and none/few of the negative examples are instances of it. The primary purpose of learning is to find a class expression, which can classify unseen

¹<http://www.w3.org>

²<http://www.w3.org/2004/OWL/>

³http://www.w3.org/TR/owl2-syntax/#Class_Expressions

individuals (i.e. not belonging to the examples) correctly. It is also important that the obtained class expression is easy to understand for a domain expert. We call these criteria *accuracy* and *readability*.

As an example, consider the problem to find out whether a chemical compound can cause cancer⁴. In this case, the background knowledge contains information about chemical compounds in general and certain concrete compounds we are interested in. The positive examples are those compounds causing cancer, whereas the negative examples are those compounds not causing cancer. The prediction for the examples has been obtained from experiments and long-term research trials in this case. Of course, all examples have to be described in the considered background knowledge. A learning algorithm can now derive a class expression from examples and background knowledge, e.g. such a class expression in natural language could be “chemical compounds containing a phosphorus atom”. (Of course, in practice the expression will be more complex to obtain a reasonable accuracy.) Using this class expression, we can not classify unseen chemical compounds.

B.2 Getting Started

DL-Learner is written in Java, i.e. it can be used on almost all platforms. Currently, Java 6 or higher is required. To install the latest release, please visit the download page⁵ and extract the file on your harddisk. In the top level directory, you will notice several executables. Those files ending with `bat` are Windows executables, whereas the corresponding files without file extension are the Non-Windows (e.g. Linux, Mac) executables. To test whether DL-Learner works, please run the following on the command line depending on your operating system:

```
dllearner examples/father.conf      (Non-Windows Operating System)
dllearner.bat examples/father.conf  (Windows Operating System)
```

Conf files, e.g. `examples/father.conf` in this case, describe the learning problem and specify which algorithm you want to use to solve it. In the simplest case they just say where to find the background knowledge to use (in the OWL file `examples/father.owl` in this case) and the positive and negative examples (marked by “+” and “-”, respectively). When running the above command, you should get something similar to the following:

```
DL-Learner 2009-05-06 command line interface
starting component manager ... OK (157ms)
initialising component "OWL file" ... OK (0ms)
initialising component "fast instance checker" ... OK (842ms)
initialising component "pos neg learning problem" ... OK (0ms)
```

⁴see <http://dl-learner.org/wiki/Carcinogenesis> for a more detailed description

⁵http://sourceforge.net/project/showfiles.php?group_id=203619

```
initialising component "refinement operator based
  learning algorithm II" ... OK (14ms)

starting top down refinement with: Thing (50% accuracy)
more accurate (83,33%) class expression found: male
solutions (at most 20 are shown):
1: (male and hasChild some Thing) (accuracy 100%, length 5, depth 3)
Algorithm terminated successfully.

number of retrievals: 4
retrieval reasoning time: 0ms (0ms per retrieval)
number of instance checks: 93 (0 multiple)
instance check reasoning time: 1ms ( 0ms per instance check)
overall reasoning time: 1ms (11,016% of overall runtime)
overall algorithm runtime: 17ms
```

The first part of the output tells you which components are used (more on this in Section B.4). In the second part you see output coming from the used learning algorithm, i.e. it can print information while running (“more accurate (83,33%) class description found”) and the final solutions, it computed. The results are displayed in Manchester OWL Syntax⁶. There can be several solutions, in which case they are ordered with the most promising one in the first position. In this case the only solution is `male and hasChild some Thing` defining the class `father`. The last part of the output contains some runtime statistics.

B.3 DL-Learner Architecture

DL-Learner consists of core functionality, which provides Machine Learning algorithms for solving the learning problem, support for different knowledge base formats, an OWL library, and reasoner interfaces. There are several interfaces for accessing this functionality, a couple of tools which use the DL-Learner algorithms, and a set of convenience scripts. The general structure is illustrated in Figure B.1.

To be flexible in integrating new learning algorithms, new kinds of learning problems, new knowledge bases, and new reasoner implementations, DL-Learner uses a component based model. Adding a component can be done by subclassing the appropriate class and adding the name of the new class to the “components.ini” file (more on that in Section B.6).

There are four types of components (knowledge source, reasoning service, learning problem, learning algorithm). For each type, there are several implemented components and each component can have its own configuration options as illustrated in Figure B.2. Configuration options can be used to change param-

⁶http://www.co-ode.org/resources/reference/manchester_syntax/

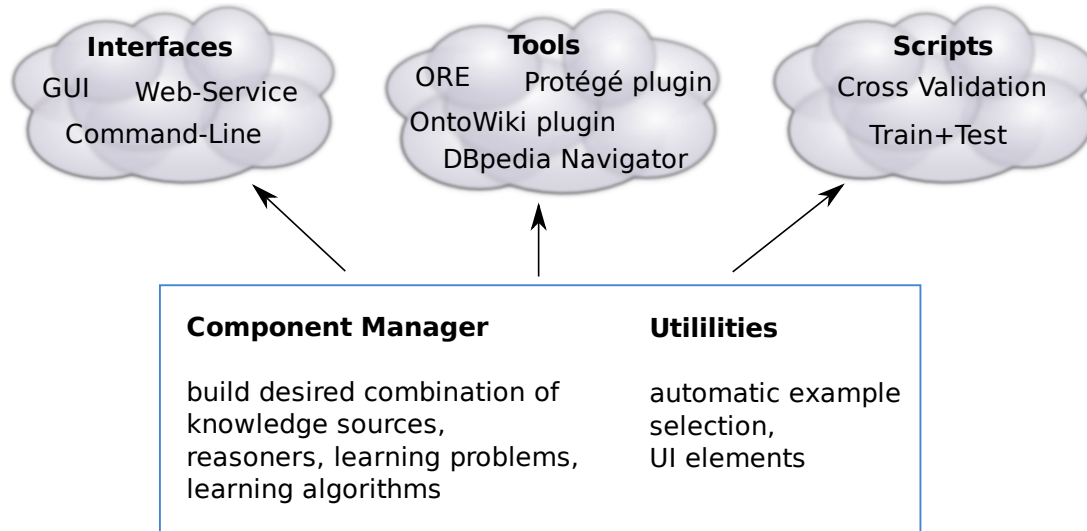


Figure B.1: Overall structure of the DL-Learner software.

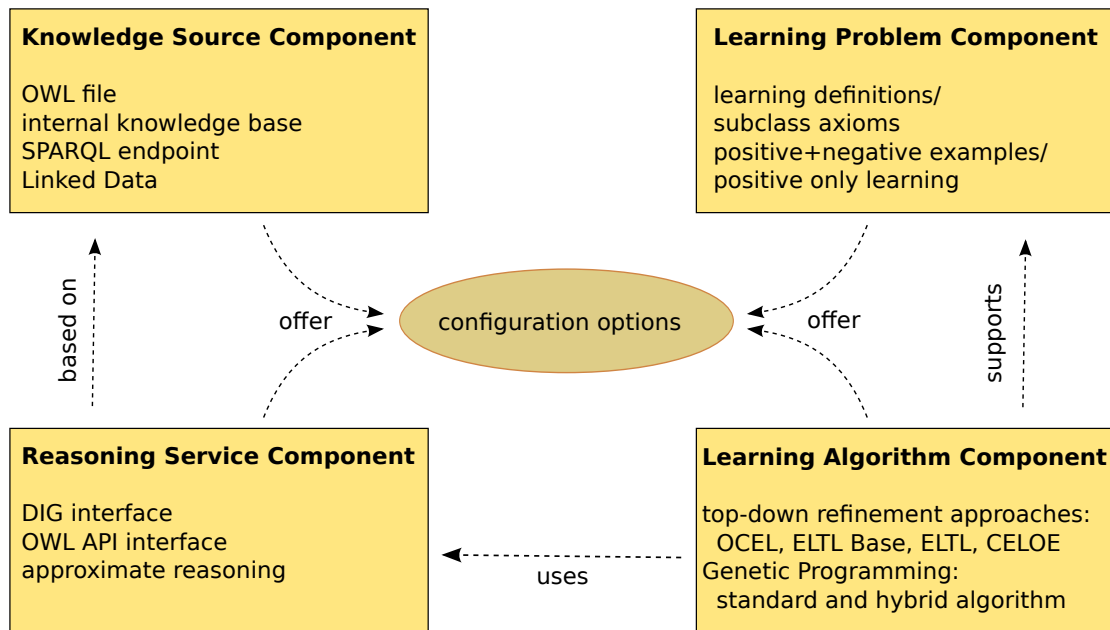


Figure B.2: The architecture of DL-Learner is based on four component types, which can each have their own configuration options. DL-Learner uses a component manager to organise all components.

eters/settings of a component. In Section B.4, we describe the components in DL-Learner and their configuration options.

B.4 DL-Learner Components

In this part, we describe concrete components currently implemented in DL-Learner. Each of the subsections contains a list of components according to the type specified in the subsection heading. Note that this does not constitute a full description, i.e. we omit some components and many configuration options. The purpose of the manual is to obtain a general understanding of the implemented components. A full list, which is generated automatically from the source code, can be found in `doc/configOptions.txt` including the default values for all options and their usage in conf files.

B.4.1 Knowledge Sources

Knowledge sources have a URI and can be included in configuration files using `import("$url");`, e.g. `import("ontology.owl")`. Depending on the file ending, DL-Learner will guess the correct type of knowledge source. If you want to overwrite this, you can use a second parameter with value `OWL`, `KB`, or `SPARQL`, e.g. `import("ontology.owl", "OWL")`.

OWL File DL-Learner supports OWL files in different formats, e.g. RDF/XML or N-Triples. If there is a standard OWL format, you want to use, but is not supported by DL-Learner please let us know. We use the OWL API for parsing, so all formats supported by it can be used⁷.

KB File KB files are an internal non-standardised knowledge representation format, which corresponds to description logic syntax except that the special symbols have been replaced by ASCII strings, e.g. `AND` instead of \sqcap . You can find several KB files in the examples folder. The `doc/kbFileSyntax.txt` contains an EBNF description of the language.

SPARQL Endpoint DL-Learner allows to use SPARQL endpoints as background knowledge source, which enables the incorporation of very large knowledge bases, e.g. DBpedia[Lehmann et al., 2009], in DL-Learner. This works by using a set of start instances, which usually correspond to the examples in a learning problem, and then retrieving knowledge about these instances via SPARQL queries. The obtained knowledge base fragment can be converted to OWL and consumed by a reasoner later since it is now sufficiently small to be processed in reasonable time. Please see [Hellmann et al., 2009a] for details about the knowledge fragment extraction algorithm. Some options of the SPARQL component are:

- **instances:** Set of individuals to use for starting the knowledge fragment extraction. Example use in conf file:

⁷ for a list see <http://owlapi.sourceforge.net>


```
sparql.instances={"http://dbpedia.org/resource/Matt_Stone",
  "http://dbpedia.org/resource/Sarah_Silverman"};
```

- `recursionDepth`: Maximum distance of an extracted individual from a start individual. This influences the size of the extracted fragment and depends on the maximum property depth you want the learned class expression to have. Example use in conf file:

```
sparql.recursionDepth = 2;
```
- `saveExtractedFragment`: Specifies whether the extracted ontology is written to a file or not. If set to true, then the OWL file is written to the cache dir. Example usage: `sparql.saveExtractedFragment = true;`

Many further options allow to modify the extracted fragment on the fly or fine-tune the extraction process. The extraction can be started separately by running and modifying `org.dllearner.test.SparqlExtractionTest`. The collected ontology will be saved in the DL-Learner directory.

B.4.2 Reasoner Components

Several reasoner components are implemented, which can be interfaces to concrete reasoners or own reasoner implementations. To select a component in a conf file, use `reasoner=$value;`, where `$value` is one of `digReasoner`, `owlAPIReasoner`, or `fastInstanceChecker`, which are explained below.

OWL API The OWL API reasoner interface can be used in conjunction with the Pellet and FaCT++ reasoners. The only option allows to switch between both:

- `reasonerType`: Selects the desired reasoner. By default, Pellet is used. Usage: `owlAPIReasoner.reasonerType = fact;`. Note that FaCT++ is written in C++ and we currently ship the 32 bit version of the JNI layer. This may change to 64 bit in the future.

DIG DIG 1.1⁸ is an interface to description logic reasoners and supported by a large variety of reasoners including Pellet, FaCT++, KAON2, and Racer Pro. The major drawback is that the current version DIG 1.1 is not aligned with the OWL specification and therefore lacks several features, which are crucial to the more recent learning algorithms in DL-Learner. If you still want to use the DIG interface, you have to download a DIG capable reasoner and start the DIG server there. DL-Learner communicates with the reasoner using the XML based protocol over HTTP.

Fast Instance Checker Instance checks, i.e. testing whether an individual is instance of a class, is the major reasoner task in many learning algorithms.

⁸<http://dl.kr.org/dig/>

This reasoner is a self-development of the DL-Learner project. It remedies some problems related to Machine Learning and the Open World Assumption in OWL and therefore is not correct w.r.t. OWL semantics. (See [Badea and Nienhuys-Cheng, 2000] Section 4 for an explanation.) Furthermore, it provides an improved performance for instance checks by precomputing some inferences and keeping them in memory. The fast instance checker is build on top of Pellet and the default reasoner component in DL-Learner.

B.4.3 Learning Problems

In the introductory Sections B.1 and B.2, we described a specific learning problem where positive and negative examples are given. In practice different variations of similar problems occur. You can switch between the different problems using `problem=$value;`, where `$value` is one of `posNegLPStandard`, `posOnlyLP`, `classLearning`. The default is `posNegLPStandard`.

Positive and Negative Examples Let the name of the background ontology be \mathcal{O} . The goal in this learning problem is to find an OWL class expression C such that all/many positive examples are instances of C w.r.t. \mathcal{O} and none/few negative examples are instances of C w.r.t. \mathcal{O} . As explained previously, C should be learned such that it generalises to unseen individuals and is readable. The important configuration options of this component are obviously the positive and negative examples, which are often indicated with $+$ and $-$ signs in conf files as an optional shortcut to using e.g. `posNegLPStandard.positiveExamples = {...}`.

Positive Examples This learning problem is similar to the one before, but without negative examples. In this case, it is desirable to find a class expression which closely fits the positive examples while still generalising sufficiently well. For instance, you usually do not want to have `owl:Thing` as a solution for this problem, but neither do you want to have an enumeration of all examples.

Class Learning In class learning, you are given an existing class A within your ontology \mathcal{O} and want to describe it. It is similar to the previous problem in that you can use the instances of the class as positive examples. However, there are some differences, e.g. you do not want to have A itself as a proposed solution of the problem, and since this is an ontology engineering task, the focus on short and readable class expressions is stronger than for the two problems mentioned before. The learner can also take advantage of existing knowledge about the class to describe.

B.4.4 Learning Algorithms

The implemented algorithms vary from very simple (and usually inappropriate) algorithms to sophisticated ones. You can switch between the different algorithms using `algorithm=$value;`, where `$value` is one of `bruteForce`, `random`, `gp`, `refinement`, `refinement2`, and `celoe`. The default is `refinement2`.

Brute Force : This algorithm tests all class expressions up to a specified length, which you can set using e.g. `bruteForce.maxlength = 7`.

Random Guesser : This algorithm randomly generates class expressions. To do this, it creates trees, which can be mapped to class expressions. Its main parameter is the number of created trees, which you can set using e.g. `random.numberOfTrees = 5`.

Genetic Programming (GP) : GP is a well-known general problem solution method, which can be adapted to class expression learning. The adaption is straightforward. In DL-Learner, however, an additional genetic refinement operator was implemented, which has shown to improve GP performance in [Lehmann, 2007]. Some options are:

- number of individuals: The individual count is the size of each generation in a GP algorithm. It is one of the most crucial parameters. Setting it to a higher value usually means investing more computational resource for increasing the likelihood that a solution will be found. Usage: `gp.numberOfIndividuals = 100`.
- refinement probability: This is used to specify how likely the usage of the genetic refinement operator should be, for instance a setting of `gp.refinementProbability = 0.6` means that it will be selected 60% of the time.

The Genetic Programming algorithm has 15 more options, which are documented in `doc/configOptions.txt`.

Refinement This is a top down refinement operator approach, which is described in [Lehmann and Hitzler, 2007c]. Some options include:

- target language: The standard target language of this algorithm is $\mathcal{ALCN}(\mathcal{D})$. However, you can change the language, i.e. you can exclude the \forall constructor by using `refinement.useAllConstructor = false;`. Similar options exist for \exists , \neg , cardinality restrictions, and boolean datatypes.
- maximum execution time: If there is no perfect solution of a given problem, the algorithm can potentially run forever (in practice it will run out of memory). It is therefore often interesting to limit the execution time. You can use e.g. `refinement.maxExecutionTimeInSeconds = 100` to say that the algorithm should run for at most 100 seconds. Often, it

will run slightly longer than the maximum execution time since it waits for the next internal loop of the algorithm to stop gracefully.

The algorithm supports a range of further options. For instance, one can specify which classes and properties must not occur in resulting class expressions.

Refinement II The previous algorithm has been extended to make more sophisticated use of background knowledge and therefore run more efficiently on many problems. It also supports double datatypes and hasValue restrictions (which again can be turned on or off as desired). It also includes explicit noise handling through the `noisePercentage` option. This is currently the default and recommend algorithm for learning from positive and negative examples. More than 30 options can be set to control its behaviour. However, apart from the target language the most important setting is noise, which should be optimised for the given problem.

Class Expression Learning for Ontology Engineering (CELOE) Currently, the CELOE algorithm is the most suitable learner for the class learning problem within DL-Learner. It uses the same refinement operator as Refinement II, but a completely different heuristics. Furthermore, it guarantees that the returned class expressions are minimal in the sense that one cannot remove parts of them without getting an inequivalent expression. Furthermore, it makes use of existing background knowledge in coverage checks. Statistical methods are used to improve the efficiency of the algorithm such that it scales to large knowledge bases. While it was originally designed for ontology engineering, it can also be used for other learning problems and might even be superior to the other algorithms in many cases (not well-tested yet). Note that many configuration options of Refinement II were dropped for the sake of simplicity, but might be introduced if needed.

Remark B.1 (Additional Algorithms)

Refinement II will be renamed to OCEL in the next release and new algorithms based on a refinement operator for the \mathcal{EL} description language will be made available. □

Please note that while components are interchangeable, it is not possible to arbitrarily combine them. For instance, the newer learning algorithms do not work with the DIG interface, since it does not provide the necessary inference tasks. Furthermore, a learning algorithm can specify which learning problems it can solve, i.e. we do not require it to be able to solve each learning problem. Table B.1 provides a compatibility matrix. Note that this can change in future releases, because algorithms may be extended to support new learning problems or drop support for them.

learning problem	BF	RG	GP	Ref	Ref II	CELOE
pos only	x	x				x
pos neg	x	x	x	x	x	x
class learning	x	x				x

Table B.1: Learning problem - learning algorithm compatibility matrix in DL-Learner. Legend: BF = brute force, RG = random guesser, Ref = Refinement

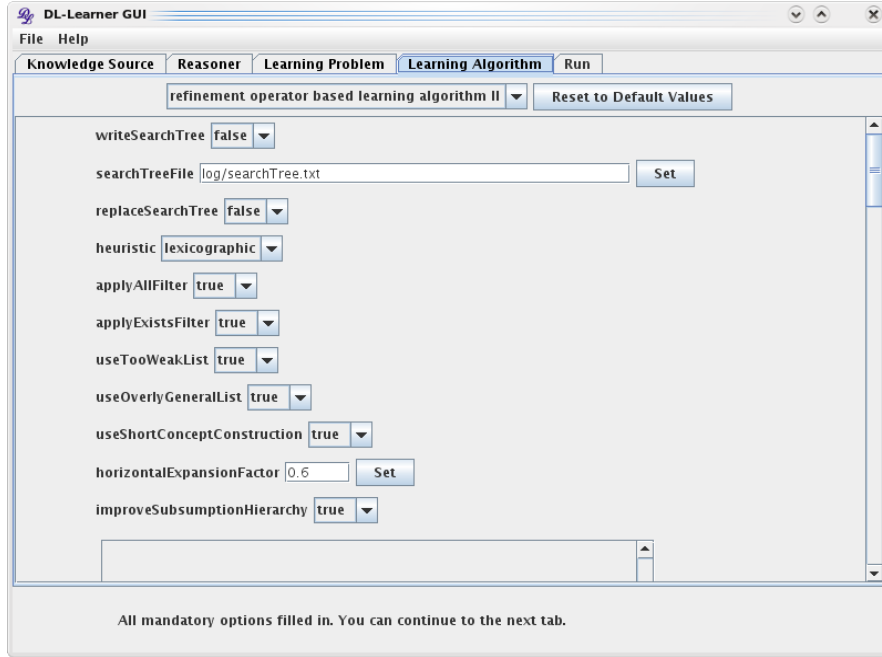


Figure B.3: GUI screenshot showing the learning algorithm tab. The UI allows you to set different options and then proceed to the next tab and execute the algorithm.

B.5 DL-Learner Interfaces

One interface you have already used in Section B.2 is the command line. There are two executables, which can be used for starting DL-Learner on the commandline: `dl-learner` and `quickstart`. The first one takes a conf file as argument, whereas the latter one lists all conf files in the examples folder and allows you to select one of those.

Apart from the command line, there is also a prototypical graphical interface. You can use `gui` (or `gui.bat`) to start it. Optionally, a conf file can be passed as argument. The main GUI window has four tabs corresponding to the four different types of components and a run tab to execute the learning algorithm. Using the GUI, you can assemble the desired combination of components and options. The **File** menu allows you to load a conf file or save the current configuration to

a conf file. The GUI implementation is currently prototypical, so please report any bugs or feature requests you have (see Section B.7). Since the GUI uses the component manager, it will automatically evolve when new components and options are added.

A third interface through which DL-Learner can be accessed programmatically is a web service. You can execute `ws` (or `ws.bat`) to start the web service. It is based on the Java API for XML Web Services (JAX-WS), which is included in Java 6 or higher. Executing the command will start a web server on port 8181 of your local machine. The WSDL can be accessed via `http://localhost:8181/services?wsdl`. You can use a WSDL viewer to see the supported operations or view the JavaDoc of the corresponding Java file⁹. Some examples for calling the web service from PHP can be found in the DL-Learner subversion repository¹⁰.

Another means to access DL-Learner, in particular for ontology engineering, is to use the OntoWiki and Protégé plugins. The OntoWiki plugin is not officially released yet, but can be used in the SVN version of OntoWiki. The Protégé 4 plugin can be installed either by downloading it from the DL-Learner download page or directly within Protégé 4 by clicking on “File”, “Preferences”, “Plugins”, “Check for Downloads” now and selecting the DL-Learner plugin. For more information and a screencast see the Protégé plugin wiki page¹¹.

B.6 Extending DL-Learner

DL-Learner is open source and component based. If you want to develop a specific part or extension of a class expression learning algorithm for OWL, then you are invited to use DL-Learner as a base. This allows you to focus on the part you want to implement while being able to use DL-Learner as a library and access it through one of the interfaces.

If you want to create a new component, then you first have to decide on the type of your component. To implement a concrete component, you have to subclass one of the following classes and implement their abstract methods:

- `org.dllearner.core.KnowledgeSource`
- `org.dllearner.core.ReasonerComponent`
- `org.dllearner.core.LearningProblem`
- `org.dllearner.core.LearningAlgorithm`

⁹viewable online at <http://dl-learner.org/javadoc/org/dllearner/server/DLLearnerWS.html>

¹⁰in the directory `src/php-examples/`:
<http://dl-learner.svn.sourceforge.net/viewvc/dl-learner/trunk/src/php-examples/>

¹¹<http://dl-learner.org/wiki/ProtegePlugin>

You then have to add your component to `lib/components.ini` such that it is registered in the component manager when DL-Learner starts up. If you want to use configuration options in your component, you need to create a static method as follows:

```
public static Collection<ConfigOption<?>> createConfigOptions() {
    List<ConfigOption<?>> options = new LinkedList<ConfigOption<?>>();
    options.add(new IntegerConfigOption("maxDepth",
        "maximum depth of generated concepts/trees", 5));
    return options;
}
```

This creates an option with name `maxDepth`, the given description, and a default value of 5. To add further options, simply add more of them to the collection. If desired, running `org.dllearner.scripts.ConfigJavaGenerator` generates a file for you in package `org.dllearner.configurators` to access the options of your component programmatically. These configurator classes are particularly useful to build scripts or tools on top of DL-Learner components. An example for this can be found in `org.dllearner.scripts.NewSample`.

Currently, the following configuration option types exist (new ones can be implemented if necessary):

- boolean, e.g. `useCache`
- string (a set of allowed strings can be specified), e.g. `cacheDir`
- URL, e.g. `reasonerURL`
- int (min and max value can be specified), e.g. `maxDepth`
- double (min and max value can be specified), e.g. `noisePercentage`
- set of strings, e.g. `positiveExamples`
- list of string tuples, e.g. `replaceObject`

Restricting to these option types this gives us the possibility to build very flexible user interfaces. Whenever, a new component or a new configuration option for a component is added, the current user interfaces (GUI, web service, commandline) will automatically support it without any or only minimal code changes.

This quick introduction only serves as an entry point to get you started. For more detailed questions about how to extend DL-Learner, please drop us a message in the DL-Learner mailing list.

B.7 General Information

- Homepage: <http://dl-learner.org>
- Sourceforge.net project page:
<http://sourceforge.net/projects/dl-learner/>
- Tracker (bugs, features):
http://sourceforge.net/tracker/?group_id=203619
- Mailing Lists: http://sourceforge.net/mail/?group_id=203619
- Contact: lehmann@informatik.uni-leipzig.de (please use the mailing list if possible)
- Latest Release:
http://sourceforge.net/project/showfiles.php?group_id=203619

C Curriculum Vitae

Personal Data

Birthdate:	March 29, 1982
Birthplace:	Meissen, Saxony, Germany
Nationality:	German
Family status:	unmarried

School Education

1992–1998	St.-Afra-Gymnasium Meissen
1998–2000	Werner-Heisenberg-Gymnasium Riesa with extended mathematical and scientific education
2000	Abitur, grade 1,2

Military Service, Study Preparations

2000–2001	Military Service at the General-Olbricht-Kaserne in Leipzig
2001	5 month practical work at System Service Meissen, (Web Applications, Scripting Languages, Databases)

Scientific Education

2001-2006	Study of Computer Science at TU Dresden
2005-2006	Erasmus exchange student at the University of Bristol (England), 4 master courses (each with grade "A") with focus on Machine Learning
2006	Diploma in Computer Science, grade 1,1
since 2006	PhD student at University of Leipzig

Miscellaneous

1993–2000	successful participation at several Math Olympics: 1st (twice), 2nd (three times), 3rd (twice) price regional level, 2nd (once), 3rd (three times) price state level (Saxony), certificate of successful participation on country level (Germany), 7th and 9th place in team competition of special math schools
1998	fee language course (4 weeks á 20 hours) in Brighton, England language diploma: grade "Advanced", level "Excellent"
1999	participation in German delegation at the "International Historical Forum of Youth", a 9 day meeting to support the understanding among nations in Krzyzowa and Opole (Poland)
2001	2 weeks Computer Science course (given by "Bildungswerk der Sächsischen Wirtschaft")

C.1 Related Peer Reviewed Publications

- ▷ Lehmann, J. and Hitzler, P. (2010). Concept learning in description logics using refinement operators. *Machine Learning journal*, 78(1-2):203–250.
- ▷ Lehmann, J. (2009). DL-Learner: learning concepts in description logics. *Journal of Machine Learning Research (JMLR)*, 10:2639–2642.
- ▷ Lehmann, J., Bizer, C., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009). DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165.
- ▷ Lehmann, J. and Haase, C. (2009). Ideal downward refinement in the \mathcal{EL} description logic. In *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium*.
- ▷ Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., and Stegemann, T. (2009). RelFinder: Revealing relationships in RDF knowledge bases. In *Proceedings of the 3rd International Conference on Semantic and Media Technologies (SAMT)*, volume 5887 of *Lecture Notes in Computer Science*, pages 182–187. Springer.
- ▷ Hellmann, S., Lehmann, J., and Auer, S. (2009a). Learning of OWL class descriptions on very large knowledge bases. *Int. J. Semantic Web Inf. Syst.*, 5(2):25–48.

- ▷ Hellmann, S., Stadler, C., Lehmann, J., and Auer, S. (2009b). DBpedia Live Extraction. In *Proc. of 8th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, volume 5871 of *Lecture Notes in Computer Science*, pages 1209–1223. Springer.
- ▷ Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2008). DBpedia: A nucleus for a web of open data. In *Proceedings of the 6th International Semantic Web Conference (ISWC)*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer.
- ▷ Hellmann, S., Lehmann, J., and Auer, S. (2008). Learning of OWL class descriptions on very large knowledge bases. In Bizer, C. and Joshi, A., editors, *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008), Karlsruhe, Germany, October 28, 2008*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- ▷ Auer, S. and Lehmann, J. (2007). What have Innsbruck and Leipzig in common? extracting semantics from wiki content. In *Proceedings of the ESWC (2007)*, LNCS (4519), pages 503–517. Springer.
- ▷ Lehmann, J. (2007). Hybrid learning of ontology classes. In Perner, P., editor, *Machine Learning and Data Mining in Pattern Recognition, 5th International Conference, MLDM 2007, Leipzig, Germany, July 18-20, 2007, Proceedings*, volume 4571 of *Lecture Notes in Computer Science*, pages 883–898. Springer.
- ▷ Lehmann, J., Schüppel, J., and Auer, S. (2007). Discovering unknown connections - the dbpedia relationship finder. In *Proceedings of the 1st SABRE Conference on Social Semantic Web*.
- ▷ Lehmann, J. and Hitzler, P. (2007a). Foundations of refinement operators for description logics. In Blockeel, H., Ramon, J., Shavlik, J. W., and Tadepalli, P., editors, *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007*, volume 4894 of *Lecture Notes in Computer Science*, pages 161–174. Springer. **Best Student Paper Award.**
- ▷ Lehmann, J. and Hitzler, P. (2007b). A refinement operator based learning algorithm for the alc description logic. In Blockeel, H., Ramon, J., Shavlik, J. W., and Tadepalli, P., editors, *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007*, volume 4894 of *Lecture Notes in Computer Science*, pages 147–160. Springer. **Best Student Paper Award.**

C.2 Other Publications

- ▷ Auer, S., Lehmann, J., and Hellmann, S. (2009c). LinkedGeoData - adding a spatial dimension to the web of data. In *Proc. of 7th International Semantic Web Conference (ISWC)*, volume 5823 of *Lecture Notes in Computer Science*, pages 731–746. Springer.
- ▷ Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., and Aumüller, D. (2009). Triplify: light-weight linked data publication from relational databases. In Quemada, J., León, G., Maarek, Y. S., and Nejd, W., editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 621–630. ACM.
- ▷ Auer, S., Lehmann, J., and Bizer, C. (2009). Semantische mashups auf basis vernetzter daten. In Blumauer, A. and Pellegrini, T., editors, *Social Semantic Web*, X.media.press, pages 259–286. Springer.
- ▷ Lehmann, J. and Haase, C. (2009). Ideal downward refinement in the \mathcal{EL} description logic. Technical report, University of Leipzig. Downloadable from <http://www.jens-lehmann.org>.
- ▷ Lehmann, J. and Knappe, S. (2008). DBpedia Navigator. Semantic Web Challenge, International Semantic Web Conference 2008.
- ▷ Lehmann, J., Bader, S., and Hitzler, P. (2008). Extracting reduced logic programs from artificial neural networks. *Applied Intelligence*.
- ▷ Riechert, T., Lauenroth, K., Lehmann, J., and Auer, S. (2007). Towards semantic based requirements engineering. In *Proceedings of the 7th International Conference on Knowledge Management (I-KNOW)*.
- ▷ Auer, S., Dietzold, S., Lehmann, J., and Riechert, T. (2007). Ontowiki: A tool for social, semantic collaboration. In Noy, N. F., Alani, H., Stumme, G., Mika, P., Sure, Y., and Vrandečić, D., editors, *Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at the 16th International World Wide Web Conference (WWW2007) Banff, Canada, May 8, 2007*, volume 273 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- ▷ Riechert, T., Lauenroth, K., and Lehmann, J. (2007). Semantisch unterstütztes requirements engineering. In *Proceedings of the SABRE-07 Software Wiki Workshop*.
- ▷ Lehmann, J. and Hitzler, P. (2007). A refinement operator based learning algorithm for the \mathcal{ALC} description logic. Technical report, University of Leipzig. Downloadable from <http://www.jens-lehmann.org>.

- ▷ Lehmann, J. and Hitzler, P. (2007). Foundations of refinement operators for description logics. Technical report, University of Leipzig. Downloadable from <http://www.jens-lehmann.org>.

C.3 Talks

- ▷ Invited speaker KYOTO EU project workshop presentation on "Semantic Wikis and the Web of Data"
- ▷ Invitation to AIFB Karlsruhe for presenting the AKSW research group and the DL-Learner project
- ▷ International Conference on Inductive Logic Programming (ILP) 2009 presentation on "Ideal Downward Refinement in the EL Description Logic"
- ▷ Two blocks (180 minutes) in the lecture "Semantic Web" 2009
- ▷ OntoWiki EU project kickoff meeting presentation on "OWL and Description Logics" 2008 and OntoWiki EU project midterm meeting presentation on "Integration of OntoWiki and DL-Learner" 2009
- ▷ International Conference on Machine Learning and Data Mining 2007 presentation on "Hybrid Learning of Ontology Classes"
- ▷ International Conference on Inductive Logic Programming (ILP) 2007 presentation on "*ALC* Concept Learning with Refinement Operators"

C.4 Research Projects and Groups

- ▷ Leader of MOLE ("Machine Learning and Ontology Engineering") research group as sub group of AKSW [since 2009]
- ▷ Participation in BmBF funded project LE4SW ("Leipzig for Semantic Web") [since 2009]
- ▷ Member of LinkedGeoData project (adding a spatial dimension to the Semantic Web) [since 2009]
- ▷ Participation in EU FP7 project "OntoWiki - Semantic Collaboration for Enterprise Knowledge Management, E-Learning and E-Tourism" [since 2008]
- ▷ Member of AKSW ("Agile Knowledge Engineering and Semantic Web") research group [since 2006]
- ▷ Member and co-founder of DBpedia open source project (extraction of machine understandable content from Wikipedia) [since 2007]

- ▷ Leader of DL-Learner open source project (framework for supervised Machine Learning in OWL and Description Logics) [since 2007]
- ▷ Member of OntoWiki open source project (agile, distributed knowledge engineering) [since 2006]
- ▷ Participation in BmBF funded SoftWiki project (distributed, end-user centered requirements engineering for evolutionary software development) [2006 - 2009]

C.5 Programm Committee, Reviewing

- ▷ PC member Workshop on Scripting and Development for the Semantic Web (SFSW) 2009
- ▷ PC member Workshop on Inductive Reasoning and Machine Learning on the Semantic Web (IRMLLeS) 2009
- ▷ PC member Workshop on Web Semantics 2009
- ▷ PC member European Semantic Web Conference 2009 (ESWC) 2009
- ▷ Reviewer European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD) 2009
- ▷ Reviewer Journal of Information Technology Research (JITR) 2009
- ▷ Reviewer Journal of Web Semantics 2007 - 2009
- ▷ Reviewer IEEE International Conference on Semantic Computing 2009
- ▷ Reviewer International Semantic Web Conference (ISWC) 2009
- ▷ Reviewer International Journal of Metadata, Semantics and Ontologies (IJMSO) 2008
- ▷ PC member I-Semantics 2008
- ▷ PC member Workshop on Scripting for the Semantic Web 2008
- ▷ Reviewer IEEE International Conference on Web Services (ICWS) 2008
- ▷ Reviewer Workshop on Linked Data on the Web 2008
- ▷ PC member SABRE Conference on Social Semantic Web (CSSW) 2007
- ▷ Reviewer European Semantic Web Conference (ESWC) 2007
- ▷ Reviewer International Joint Conference on Neural Networks (IJCNN) 2007

C.6 Seminars and Teaching

- ▷ Assistant lecturer for “Semantic Web”, 2009
- ▷ Practical course “Semantic Web”, 2009
- ▷ Seminar “Softwareproduktlinienentwicklung und Semantic Web”, 2008/09
- ▷ Seminar “Semantische Unterstützung von Software Entwicklungsprozessen”, 2008
- ▷ Seminar “Semantische Unterstützung von Software Entwicklungsprozessen”, 2007/08
- ▷ Seminar “Semantic Web Applications”, 2007
- ▷ Seminar “Semantic Web Services and Interfaces”, 2006/07
- ▷ Research seminar of the chair of Business Information Systems, since 2006

C.7 Supervision

- ▷ Steffen Becker, diploma thesis, “Musikempfehlungen im Semantic Web”, 2009
- ▷ Lorenz Bühmann, master thesis, “Ontology Repair and Enrichment”, 2009
- ▷ Vu Duc Minh, bachelor thesis, “DL-Learner Plugin für OntoWiki”, 2008
- ▷ Lorenz Bühmann, bachelor thesis, “Erweiterung und Reparatur von Ontologien”, 2008
- ▷ Christian Kötteritzsch, bachelor thesis, “Entwicklung eines DL-Learner Plugins für Protégé”, 2008
- ▷ Maria Moritz, bachelor thesis, “DL-Learner PlugIn für OntoWiki”, 2008 (with Sebastian Dietzold)
- ▷ Sebastian Knappe, diploma thesis, “Navigation in DBpedia mit Hilfe maschinellen Lernens” ,2007/2008
- ▷ Tilo Hielscher, bachelor thesis, “Entwicklung einer Java-basierten Oberfläche für das DL-Learner-Tool” ,2007/2008
- ▷ Ruslan Masold, bachelor thesis, “Verwendung von Benutzeroberflächen mit Web 2.0 Technologien in einer semantischen Webapplikation”, 2007/2008
- ▷ Tom Weiland, bachelor thesis, “Controller Design für semantische Webapplikationen”, 2007/2008

- ▷ Rolland Brunec, bachelor thesis, “Extending OntoWiki with a Manchester OWL Syntax Parser”, 2007/2008
- ▷ Sebastian Hellmann, diploma thesis, “Comparison of Concept Learning Algorithms”, 2007/2008
- ▷ Fan Zhang, bachelor thesis, “Internationalisierung und Usability von Ontologie-Editoren am Beispiel OntoWiki”, 2007
- ▷ Anke Gonschorreck, bachelor thesis, “Extraktion einer Klassenhierarchie basierend auf den Wikipedia-Kategorien”, 2007
- ▷ Jörg Schüppel, diploma thesis, “Semantikextraktion aus Wikipedia”, 2006/2007 (with Sören Auer)

In addition, the work of many student assistants was supervised.

List of Tables

1.1	Contributions and advancements over the state of the art.	17
2.1	<i>SHOIN</i> Syntax and Semantics	26
2.2	Mapping between OWL and DL	32
4.1	Syntax and Semantics of Extensions Supported by ρ	72
4.2	\mathcal{EL} Syntax and Semantics	76
4.3	\mathcal{EL} Knowledge Base Axioms	76
4.4	\mathcal{EL} Operator Benchmark Results	88
5.1	Ontology Engineering Heuristics	101
6.1	Common DBpedia Classes	109
6.2	Comparison of Generic Infobox, Mapping-Based Infobox, and Pagelinks Datasets	111
6.3	Comparison of Graph Structures of Generic Infobox, Mapping- Based Infobox, and Pagelinks Datasets	111
6.4	Distribution of Outgoing Links of DBpedia	114
6.5	Data Sources Publishing Links to DBpedia	115
6.6	Semantic Bible Fragment Extraction Statistics	121
6.7	The table shows a probe of the automatic re-learning method for classes. Sets were evaluated manually, falsely classified individuals in brackets	131
7.1	OCEL and ELTL Results	143
7.2	YinYang and GP Results	144
7.3	Comparison with ILP Approaches on Carcinogenesis Problem . .	148
7.4	Influence of Noise Parameter	150
7.5	Ontology Engineering Evaluation Results	157
7.6	Ontology Engineering Performance Assessment	159
7.7	Comparison of Fragment Extraction and Standard Learning . . .	161
B.1	Learning Problem - Learning Algorithm Compatibility Matrix . .	189

List of Figures

2.1	Semantic Web Layer Cake	20
2.2	RDF Example Graph	21
2.3	Integration of Thesis into the Research Areas Machine Learning and Knowledge Representation	39
2.4	Generate and test approach used in DL-Learner	42
2.5	Illustration of some refinement operator properties (left to right): completeness, redundancy, properness, and finiteness (bottom). . .	44
4.1	Definition of the refinement operator ρ	63
4.2	Simulation Relation Example	78
4.3	\mathcal{EL} Refinement Chain Example	83
4.4	Application of \mathcal{EL} Refinement Operator	85
5.1	Illustration of a search tree in a top down refinement approach. .	91
5.2	Michalski Trains Problem	94
5.3	Visualisation of Accuracy Measurement Approaches	100
6.1	Overview of DBpedia Components	105
6.2	Infobox Tom Hanks.	107
6.3	Comparison of Node Indegree versus Rank of Generic Infobox, Mapping-Based Infobox, and Pagelinks Datasets	112
6.4	DBpedia Example RDF Links	113
6.5	LOD Cloud	114
6.6	Revealing relationships between Kurt Gödel und Albert Einstein. .	117
6.7	Overview of the DBpedia Navigator GUI. DBpedia Navigator pro- vides an interface for searching and browsing within DBpedia. . .	118
6.8	Fragment Extraction Process	121
6.9	Fragment Extraction Visualisation	126
7.1	Overall Structure of the DL-Learner Software	138
7.2	DL-Learner Component Types	139
7.3	Accuracy Comparison	145
7.4	Length Comparison	146
7.5	Runtime Comparison	147
7.6	Visualisation of Training and Testing Accuracy	151
7.7	Screenshot of DL-Learner Protégé Plugin	153
7.8	OntoWiki DL-Learner Plugin Invocation	154

7.9	OntoWiki DL-Learner Plugin Result Table	155
7.10	Analysis of Extracted Triples and Extraction Time	160
7.11	Time versus Accuracy Comparison for Semantic Bible Example .	162
B.1	Overall Structure of DL-Learner Software	183
B.2	DL-Learner Component Architecture	183
B.3	DL-Learner GUI Screenshot	189

List of Algorithms

1	Computing the Maximal Simulation	81
2	Computation of the Set $as(t, v)$	83
3	OCEL Algorithm	93
4	ELTL Base Algorithm	95
5	ELTL Algorithm	96
6	Knowledge Extraction Algorithm	127

List of Definitions

2.5	Syntax of \mathcal{ALC} concepts	24
2.7	Interpretation	26
2.10	Consistency	29
2.12	Satisfiability	29
2.14	Subsumption, Equivalence	29
2.16	Instance Check	29
2.17	Retrieval	30
2.19	Length of an \mathcal{ALC} Concept	30
2.20	Concept Learning with Background Knowledge	35
2.21	Learning From Examples in OWL/DLs	39
2.22	Positive Only Learning in OWL/DLs	40
2.23	Class/Concept Learning in OWL/DLs	40
2.25	Refinement Operator	42
2.26	L Refinement Operator	42
2.27	Refinement Chain	42
2.28	Downward and Upward Cover	43
2.29	Properties of DL Refinement Operators	43
4.3	S_{\downarrow}	65
4.8	ρ^{cl}	70
4.11	\mathcal{EL} Simulation	78
4.12	\mathcal{EL} Tree	78
4.13	\mathcal{EL} Graph of an Interpretation	79
4.18	Minimal Trees	81
4.21	\mathcal{EL} Refinement Operator ψ	84
5.1	OCEL Score	92
5.4	CELOE Score	101
6.2	Tuple Acquisition Interface	124

List of Theorems

3.1	Existence of Covers in \mathcal{ALC}	46
3.2	minimality and weak completeness	47
3.3	Complete and Finite Refinement Operators	49
3.5	Non-Ideality Helper Lemma 1	50
3.6	Non-Ideality Helper Lemma 2	51
3.7	Ideal Refinement Operators	52
3.8	Complete and Proper Refinement Operators	53
3.9	Complete, Non-redundant Refinement Operators	53
3.10	Complete, Non-redundant Operators II	55
3.11	Incomplete Refinement Operators	55
3.12	Properties of Refinement Operators (I)	56
3.13	Weakly Complete, Non-redundant, Proper Op.	57
3.14	Weakly Complete, Non-redundant, Finite Op.	57
3.15	Weakly Complete, Proper, and Finite Operators	58
3.16	Properties of Refinement Operators (II)	58
4.2	Downward Refinement of ρ	64
4.4	Adequacy of S_{\downarrow}	66
4.5	Weak Completeness with Domain B	67
4.6	Weak Completeness of ρ	69
4.7	Completeness of ρ	69
4.9	ρ Does not Reduce Length	71
4.10	Computability up to Length n	71
4.14	Simulations and Interpretations	79
4.15	Uniqueness of Maximal Simulations	80
4.16	Subsumption and Simulations	80
4.17	Computation of Maximal Simulation	80
4.19	Finite Number of \mathcal{EL} Trees up to some Depth	81
4.20	No Simulation Between Trees of Different Depth	82
4.23	Finiteness, Properness, Weak Completeness of ψ	84
4.24	Ideality of ψ^*	86
4.25	No Redundant and Complete \mathcal{EL} Operators	86
5.2	Correctness	94

List of Examples and Remarks

1.1	Simple Ontology Engineering Use Case	13
2.1	RDF/XML Syntax	21
2.2	RDF Turtle Syntax	22
2.3	SPARQL Query	23
2.4	Filters in SPARQL Queries	23
2.6	\mathcal{ALC} concepts	24
2.8	Interpreting Concepts	26
2.9	Models of a Knowledge Base	27
2.11	Consistency	29
2.13	Satisfiability	29
2.15	Subsumption	29
2.18	Instance Check, Retrieval	30
2.24	Concept Learning Example	40
3.4	Complete and Finite Refinement Operators	50
3.17	Subsumption with Respect to a TBox	59
3.18	Weak Equality	59
3.19	\mathcal{EL} Family of Description Logics	59
4.1	ρ refinements	64
4.22	Application of ψ	84
4.26	Application of Extended Operator ψ	87
5.3	OCEL	94
6.1	Manual Example From Semantic Bible	120
6.3	Example SPARQL Query on DBpedia	128
6.4	Re-Learning Wikipedia Categories	130
6.5	Navigation Use Case	132
6.6	Stochastic Coverage Test	136
7.1	Extended Evaluation	158
B.1	Additional Algorithms	188

Bibliography

- [Agresti and Coull, 1998] Agresti, A. and Coull, B. A. (1998). Approximate is better than “exact” for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126.
- [Auer et al., 2008] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2008). DBpedia: A nucleus for a web of open data. In *Proceedings of the 6th International Semantic Web Conference (ISWC)*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer.
- [Auer et al., 2009a] Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., and Aumueller, D. (2009a). Triplify: light-weight linked data publication from relational databases. In Quemada, J., León, G., Maarek, Y. S., and Nejdl, W., editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 621–630. ACM.
- [Auer et al., 2007] Auer, S., Dietzold, S., Lehmann, J., and Riechert, T. (2007). Ontowiki: A tool for social, semantic collaboration. In Noy, N. F., Alani, H., Stumme, G., Mika, P., Sure, Y., and Vrandecic, D., editors, *Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at the 16th International World Wide Web Conference (WWW2007) Banff, Canada, May 8, 2007*, volume 273 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- [Auer et al., 2006a] Auer, S., Dietzold, S., and Riechert, T. (2006a). OntoWiki – A tool for Social, Semantic Collaboration. In *The Semantic Web – ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer.
- [Auer et al., 2006b] Auer, S., Dietzold, S., and Riechert, T. (2006b). Ontowiki - a tool for social, semantic collaboration. In *ISWC 2006*, volume 4273 of *LNCS*, pages 736–749. Springer.
- [Auer and Lehmann, 2007] Auer, S. and Lehmann, J. (2007). What have Innsbruck and Leipzig in common? extracting semantics from wiki content. In *Proceedings of the ESWC (2007)*, LNCS (4519), pages 503–517. Springer.
- [Auer et al., 2009b] Auer, S., Lehmann, J., and Bizer, C. (2009b). Semantische mashups auf basis vernetzter daten. In Blumauer, A. and Pellegrini, T., editors, *Social Semantic Web*, X.media.press, pages 259–286. Springer.

- [Auer et al., 2009c] Auer, S., Lehmann, J., and Hellmann, S. (2009c). Linked-GeoData - adding a spatial dimension to the web of data. In *Proc. of 7th International Semantic Web Conference (ISWC)*, volume 5823 of *Lecture Notes in Computer Science*, pages 731–746. Springer.
- [Baader et al., 2007a] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2007a). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.
- [Baader et al., 2007b] Baader, F., Ganter, B., Sattler, U., and Sertkaya, B. (2007b). Completing description logic knowledge bases using formal concept analysis. In *IJCAI 2007*. AAAI Press.
- [Baader et al., 1999] Baader, F., Molitor, R., and Tobies, S. (1999). Tractable and decidable fragments of conceptual graphs. In *Seventh International Conference on Conceptual Structures (ICCS'99)*, number 1640 in LNCS, pages 480–493. Springer Verlag.
- [Baader et al., 2007c] Baader, F., Sertkaya, B., and Turhan, A.-Y. (2007c). Computing the least common subsumer w.r.t. a background terminology. *J. Applied Logic*, 5(3):392–420.
- [Badea, 2000] Badea, L. (2000). Perfect refinement operators can be flexible. In Horn, W., editor, *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 266–270. IOS Press.
- [Badea and Nienhuys-Cheng, 2000] Badea, L. and Nienhuys-Cheng, S.-H. (2000). A refinement operator for description logics. In Cussens, J. and Frisch, A., editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 40–59. Springer-Verlag.
- [Badea and Stanciu, 1999] Badea, L. and Stanciu, M. (1999). Refinement operators can be (weakly) perfect. In Džeroski, S. and Flach, P., editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 21–32. Springer-Verlag.
- [Battista et al., 2007] Battista, A. D. L., Villanueva-Rosales, N., Palenychka, M., and Dumontier, M. (2007). SMART: A web-based, ontology-driven, semantic web query answering application. In *Semantic Web Challenge at the ISWC 2007*.
- [Bechhofer and Volz, 2004] Bechhofer, S. and Volz, R. (2004). Patching syntax in OWL ontologies. In McIlraith, S. A., Plexousakis, D., and van Harmelen, F., editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 668–682. Springer.

- [Becker and Bizer, 2008] Becker, C. and Bizer, C. (2008). DBpedia Mobile - A Location-Aware Semantic Web Client. In *Proceedings of the Semantic Web Challenge*.
- [Belleau et al., 2008] Belleau, F., Tourigny, N., Good, B., and Morissette, J. (2008). Bio2RDF: A semantic web atlas of post genomic knowledge about human and mouse. In Bairoch, A., Boulakia, S. C., and Froidevaux, C., editors, *DILS*, volume 5109 of *Lecture Notes in Computer Science*, pages 153–160. Springer.
- [Benigni and Giuliani, 2003] Benigni, R. and Giuliani, A. (2003). Putting the predictive toxicology challenge into perspective: Reflections on the results. *Bioinformatics*, 19(10):1194–1200.
- [Bergadano and Gunetti, 1995] Bergadano, F. and Gunetti, D. (1995). *Inductive Logic Programming: From Machine Learning to Software Engineering*. The MIT Press.
- [Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.
- [Blockeel and Raedt, 1996] Blockeel, H. and Raedt, L. D. (1996). Relational knowledge discovery in databases. In Muggleton, S., editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 1–13. Stockholm University, Royal Institute of Technology.
- [Blumer et al., 1990] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. (1990). Occam’s razor. In Shavlik, J. W. and Dietterich, T. G., editors, *Readings in Machine Learning*, pages 201–204. Morgan Kaufmann.
- [Bodenreider et al., 2007] Bodenreider, O., Smith, B., Kumar, A., and Burgun, A. (2007). Investigating subsumption in SNOMED CT: An exploration into large description logic-based biomedical terminologies. *Artificial Intelligence in Medicine*, 39(3):183–195.
- [Boström, 1996] Boström, H. (1996). Theory-guided induction of logic programs by inference of regular languages. In *Proc. of the 13th International Conference on Machine Learning*, pages 46–53. Morgan Kaufmann.
- [Brachman, 1978] Brachman, R. J. (1978). A structural paradigm for representing knowledge. Technical Report BBN Report 3605, Bolt, Beranek and Newman, Inc., Cambridge, MA.
- [Bratko, 1999] Bratko, I. (1999). Refining complete hypotheses in ILP. In Džeroski, S. and Flach, P., editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 44–55. Springer-Verlag.

- [Buitelaar et al., 2007] Buitelaar, P., Cimiano, P., and Magnini, B., editors (2007). *Ontology Learning from Text: Methods, Evaluation and Applications*, volume 123 of *Frontiers in Artificial Intelligence*. IOS Press.
- [Cheng et al., 2008] Cheng, G., Ge, W., Wu, H., and Qu, Y. (2008). Searching Semantic Web Objects Based on Class Hierarchies. In *Proceedings of the 1st Linked Data on the Web Workshop*.
- [Cimiano et al., 2009] Cimiano, P., Rudolph, S., and Hartfiel, H. (2009). Computing intensional answers to questions - an inductive logic programming approach. *Journal of Data and Knowledge Engineering (DKE)*.
- [Clark et al., 2008] Clark, K. G., Feigenbaum, L., and Torres, E. (2008). SPARQL Protocol for RDF. W3c recommendation, W3C.
- [Cohen et al., 1993] Cohen, W. W., Borgida, A., and Hirsh, H. (1993). Computing least common subsumers in description logics. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 754–760. AAAI Press.
- [Cohen and Hirsh, 1994] Cohen, W. W. and Hirsh, H. (1994). Learning the CLASSIC description logic: Theoretical and experimental results. In Doyle, J., Sandewall, E., and Torasso, P., editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 121–133. Morgan Kaufmann.
- [Cussens, 1996] Cussens, J. (1996). Part-of-speech disambiguation using ILP. Technical Report PRG-TR-25-96, Oxford University Computing Laboratory.
- [d’Amato et al., 2005] d’Amato, C., Fanizzi, N., and Esposito, F. (2005). A semantic similarity measure for expressive description logics. In *Proceedings of the Convegno Italiano di Logica Computazionale*.
- [d’Amato et al., 2006] d’Amato, C., Fanizzi, N., and Esposito, F. (2006). Reasoning by analogy in description logics through instance-based learning. In Tummarello, G., Bouquet, P., and Signore, O., editors, *SWAP 2006 - Semantic Web Applications and Perspectives, Proceedings of the 3rd Italian Semantic Web Workshop, Scuola Normale Superiore, Pisa, Italy, 18-20 December, 2006*, volume 201 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- [d’Amato et al., 2008a] d’Amato, C., Fanizzi, N., and Esposito, F. (2008a). A note on the evaluation of inductive concept classification procedures. In Gangemi, A., Keizer, J., Presutti, V., and Stoermer, H., editors, *Proceedings of the 5th Workshop on Semantic Web Applications and Perspectives (SWAP2008), Rome, Italy, December 15-17, 2008*, volume 426 of *CEUR Workshop Proceedings*. CEUR-WS.org.

- [d’Amato et al., 2008b] d’Amato, C., Fanizzi, N., and Esposito, F. (2008b). Query answering and ontology population: An inductive approach. In *ESWC*, pages 288–302.
- [Davies et al., 2006] Davies, J., Studer, R., and Warren, P., editors (2006). *Semantic Web Technologies – trends and research in ontology-based systems*. John Wiley & Sons.
- [de Bruijn et al., 2005] de Bruijn, J., Lara, R., Polleres, A., and Fensel, D. (2005). OWL DL vs. OWL flight: conceptual modeling and reasoning for the semantic web. In Ellis, A. and Hagino, T., editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 623–632. ACM.
- [Dietterich et al., 2008] Dietterich, T., Domingos, P., Getoor, L., Muggleton, S., and Tadepalli, P. (2008). Structured machine learning: the next ten years. *Machine Learning*, 73(1):3–23.
- [Domingos, 1998] Domingos, P. (1998). Occam’s two razors: The sharp and the blunt. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 37–43.
- [Dutra et al., 2003] Dutra, I., Page, D., Costa, V. S., and Shavlik, J. (2003). An empirical evaluation of bagging in inductive logic programming. In Matwin, S. and Sammut, C., editors, *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 48–65. Springer-Verlag.
- [Džeroski et al., 1998] Džeroski, S., Jacobs, N., Molina, M., Moure, C., Muggleton, S., and Laer, W. V. (1998). Detecting traffic problems with ILP. In Page, D., editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, pages 281–290. Springer-Verlag.
- [Dzeroski et al., 1998] Dzeroski, S., Jacobs, N., Molina, M., and Moure, C. (1998). ILP experiments in detecting traffic problems. In *10th European Conference on Machine Learning*, *Lecture Notes in Artificial Intelligence*, pages 61–66. Springer-Verlag.
- [Esposito et al., 2004] Esposito, F., Fanizzi, N., Iannone, L., Palmisano, I., and Semeraro, G. (2004). Knowledge-intensive induction of terminologies from metadata. In *The Semantic Web – ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan, November 7-11, 2004. Proceedings*, pages 441–455. Springer.

- [Esposito et al., 2001] Esposito, F., Malerba, D., and Marengo, V. (2001). Inductive learning from numerical and symbolic data: An integrated framework. *Intell. Data Anal.*, 5(6):445–461.
- [Fanizzi et al., 2008] Fanizzi, N., d’Amato, C., and Esposito, F. (2008). DL-FOIL concept learning in description logics. In *Proceedings of the 18th International Conference on Inductive Logic Programming*, volume 5194 of *LNCS*, pages 107–121. Springer.
- [Fanizzi et al., 2004] Fanizzi, N., Ferilli, S., Iannone, L., Palmisano, I., and Semeraro, G. (2004). Downward refinement in the ALN description logic. In *HIS*, pages 68–73. IEEE Computer Society.
- [Fanizzi et al., 2003] Fanizzi, N., Ferilli, S., Mauro, N. D., and Basile, T. M. A. (2003). Spaces of theories with ideal refinement operators. In Gottlob, G. and Walsh, T., editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 527–532. Morgan Kaufmann.
- [Fensel et al., 2008] Fensel, D., van Harmelen, F., Andersson, B., Brennan, P., Cunningham, H., Valle, E. D., Fischer, F., Huang, Z., Kiryakov, A., il Lee, T. K., Schooler, L., Tresp, V., Wesner, S., Witbrock, M., and Zhong, N. (2008). Towards larKC: A platform for web-scale reasoning. In *ICSC*, pages 524–529. IEEE Computer Society.
- [Fokoue et al., 2006] Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., and Srinivas, K. (2006). The summary ABox: Cutting ontologies down to size. In *ISWC*, pages 343–356.
- [Harth et al., 2008] Harth, A., Hogan, A., Umbrich, J., and Decker, S. (2008). Swse: Objects before documents! In *Proceedings of the Semantic Web Challenge*.
- [Heim et al., 2009] Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., and Stegemann, T. (2009). RelFinder: Revealing relationships in RDF knowledge bases. In *Proceedings of the 3rd International Conference on Semantic and Media Technologies (SAMT)*, volume 5887 of *Lecture Notes in Computer Science*, pages 182–187. Springer.
- [Hellmann et al., 2008] Hellmann, S., Lehmann, J., and Auer, S. (2008). Learning of OWL class descriptions on very large knowledge bases. In Bizer, C. and Joshi, A., editors, *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008), Karlsruhe, Germany, October 28, 2008*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org.

- [Hellmann et al., 2009a] Hellmann, S., Lehmann, J., and Auer, S. (2009a). Learning of OWL class descriptions on very large knowledge bases. *Int. J. Semantic Web Inf. Syst.*, 5(2):25–48.
- [Hellmann et al., 2009b] Hellmann, S., Stadler, C., Lehmann, J., and Auer, S. (2009b). DBpedia Live Extraction. In *Proc. of 8th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, volume 5871 of *Lecture Notes in Computer Science*, pages 1209–1223. Springer.
- [Helma et al., 2001] Helma, C., King, R. D., Kramer, S., and Srinivasan, A. (2001). The predictive toxicology challenge 2000-2001. *Bioinformatics*, 17(1):107–108.
- [Hitzler et al., 2009] Hitzler, P., Krötzsch, M., and Rudolph, S. (2009). *Foundations of Semantic Web Technologies*. CRC Press/Chapman & Hall.
- [Horridge et al., 2008] Horridge, M., Parsia, B., and Sattler, U. (2008). Laconic and precise justifications in owl. In *ISWC 08 The International Semantic Web Conference 2008, Karlsruhe, Germany*.
- [Horridge and Patel-Schneider, 2008] Horridge, M. and Patel-Schneider, P. F. (2008). Manchester syntax for OWL 1.1. In *OWLED 2008, 4th international workshop OWL: Experiences and Directions*.
- [Horrocks et al., 2006] Horrocks, I., Kutz, O., and Sattler, U. (2006). The even more irresistible SROIQ. In Doherty, P., Mylopoulos, J., and Welty, C. A., editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 57–67. AAAI Press.
- [Horrocks et al., 2004] Horrocks, I., Li, L., Turi, D., and Bechhofer, S. (2004). The instance store: DL reasoning with large numbers of individuals. In Haarslev, V. and Möller, R., editors, *Description Logics*, volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- [Horrocks et al., 2003] Horrocks, I., Patel-Schneider, P. F., and van Harmelen, F. (2003). From *SHIQ* and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26.
- [Horvath et al., 2009] Horvath, T., Paass, G., Reichartz, F., and Wrobel, S. (2009). A logic-based approach to relation extraction from texts. In *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium*.
- [Iannone and Palmisano, 2005] Iannone, L. and Palmisano, I. (2005). An algorithm based on counterfactuals for concept learning in the semantic web. In *Proceedings of the 18th International Conference on Industrial and Engineering*

- Applications of Artificial Intelligence and Expert Systems*, pages 370–379, Bari, Italy.
- [Iannone et al., 2007] Iannone, L., Palmisano, I., and Fanizzi, N. (2007). An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159.
- [Jiang and Colton, 2006] Jiang, N. and Colton, S. (2006). Boosting descriptive ILP for predictive learning in bioinformatics. In Muggleton, S., Otero, R. P., and Tamaddoni-Nezhad, A., editors, *Proceedings of the 15th International Conference on Inductive Logic Programming*, volume 4455 of *Lecture Notes in Computer Science*, pages 275–289. Springer.
- [Kazakov, 1999] Kazakov, D. (1999). Combining LAPIS and WordNet for the learning of LR parsers with optimal semantic constraints. In Džeroski, S. and Flach, P., editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 140–151. Springer-Verlag.
- [Kersting, 2006] Kersting, K. (2006). An inductive logic programming approach to statistical relational learning. *AI Commun*, 19(4):389–390.
- [Kietz and Morik, 1994] Kietz, J.-U. and Morik, K. (1994). A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14:193–217.
- [King et al., 1995] King, R. D., Sternberg, M. J. E., and Srinivasan, A. (1995). Relating chemical activity to structure: An examination of ILP successes. *New Generation Comput*, 13(3&4):411–433.
- [Kobilarov et al., 2009] Kobilarov, G., Scott, T., Raimond, Y., Oliver, S., Sizemore, C., Smethurst, M., Lee, R., and Bizer, C. (2009). Media meets semantic web - how the bbc uses DBpedia and linked data to make connections. In *Proceedings of the 6th European Semantic Web Conference*.
- [Krötzsch et al., 2008a] Krötzsch, M., Rudolph, S., and Hitzler, P. (2008a). Description logic rules. In Ghallab, M. et al., editors, *Proceedings of the 18th European Conf. on Artificial Intelligence (ECAI-08)*, pages 80–84. IOS Press.
- [Krötzsch et al., 2008b] Krötzsch, M., Rudolph, S., and Hitzler, P. (2008b). ELP: Tractable Rules for OWL 2. In Sheth, A. et al., editors, *The Semantic Web – ISWC 2008, 7th International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 649–664. Springer.
- [Küsters, 2001] Küsters, R. (2001). *Non-standard inferences in description logics*. Springer-Verlag New York, Inc., New York, NY, USA.

- [Lamma et al., 1999] Lamma, E., Riguzzi, F., and Pereira, L. M. (1999). Learning three-valued logic programs. In Dzeroski, S. and Flach, P., editors, *ILP-99 Late-Breaking Papers*, pages 30–35.
- [Lavrac and Dzeroski, 1994] Lavrac, N. and Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Artificial Intelligence. Ellis Horwood (Simon & Schuster).
- [Leban et al., 2008] Leban, G., Zabkar, J., and Bratko, I. (2008). An experiment in robot discovery with ILP. In Zelezný, F. and Lavrac, N., editors, *Inductive Logic Programming, 18th International Conference, ILP 2008, Prague, Czech Republic, September 10-12, 2008, Proceedings*, volume 5194 of *Lecture Notes in Computer Science*, pages 77–90. Springer.
- [Lehmann, 2007] Lehmann, J. (2007). Hybrid learning of ontology classes. In Perner, P., editor, *Machine Learning and Data Mining in Pattern Recognition, 5th International Conference, MLDM 2007, Leipzig, Germany, July 18-20, 2007, Proceedings*, volume 4571 of *Lecture Notes in Computer Science*, pages 883–898. Springer.
- [Lehmann, 2009] Lehmann, J. (2009). DL-Learner: learning concepts in description logics. *Journal of Machine Learning Research (JMLR)*, 10:2639–2642.
- [Lehmann et al., 2008] Lehmann, J., Bader, S., and Hitzler, P. (2008). Extracting reduced logic programs from artificial neural networks. *Applied Intelligence*.
- [Lehmann et al., 2009] Lehmann, J., Bizer, C., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009). DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165.
- [Lehmann and Haase, 2009a] Lehmann, J. and Haase, C. (2009a). Ideal downward refinement in the EL description logic. In *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium*.
- [Lehmann and Haase, 2009b] Lehmann, J. and Haase, C. (2009b). Ideal downward refinement in the EL description logic. Technical report, University of Leipzig. Downloadable from <http://www.jens-lehmann.org>.
- [Lehmann and Hitzler, 2007a] Lehmann, J. and Hitzler, P. (2007a). Foundations of refinement operators for description logics. In Blockeel, H., Ramon, J., Shavlik, J. W., and Tadepalli, P., editors, *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007*, volume 4894 of *Lecture Notes in Computer Science*, pages 161–174. Springer. Best Student Paper Award.
- [Lehmann and Hitzler, 2007b] Lehmann, J. and Hitzler, P. (2007b). Foundations of refinement operators for description logics. Technical report, University of Leipzig. Downloadable from <http://www.jens-lehmann.org>.

- [Lehmann and Hitzler, 2007c] Lehmann, J. and Hitzler, P. (2007c). A refinement operator based learning algorithm for the \mathcal{ALC} description logic. In Blockeel, H., Ramon, J., Shavlik, J. W., and Tadepalli, P., editors, *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007*, volume 4894 of *Lecture Notes in Computer Science*, pages 147–160. Springer. Best Student Paper Award.
- [Lehmann and Hitzler, 2007d] Lehmann, J. and Hitzler, P. (2007d). A refinement operator based learning algorithm for the \mathcal{ALC} description logic. Technical report, University of Leipzig. Downloadable from <http://www.jens-lehmann.org>.
- [Lehmann and Hitzler, 2010] Lehmann, J. and Hitzler, P. (2010). Concept learning in description logics using refinement operators. *Machine Learning journal*, 78(1-2):203–250.
- [Lehmann and Knappe, 2008] Lehmann, J. and Knappe, S. (2008). Dbpedia navigator. Semantic Web Challenge, International Semantic Web Conference 2008.
- [Lehmann et al., 2007] Lehmann, J., Schüppel, J., and Auer, S. (2007). Discovering unknown connections - the dbpedia relationship finder. In *Proceedings of the 1st SABRE Conference on Social Semantic Web*.
- [Lenat, 1995] Lenat, D. (1995). CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38.
- [Lisi, 2005] Lisi, F. A. (2005). Principles of inductive reasoning on the semantic web: A framework for learning in AL-log. In Fages, F. and Soliman, S., editors, *Principles and Practice of Semantic Web Reasoning, Third International Workshop, PPSWR 2005, Dagstuhl Castle, Germany, September 11-16, 2005, Proceedings*, volume 3703 of *Lecture Notes in Computer Science*, pages 118–132. Springer.
- [Lisi, 2008] Lisi, F. A. (2008). Building rules on top of ontologies for the semantic web with inductive logic programming. *TPLP*, 8(3):271–300.
- [Lisi and Esposito, 2008] Lisi, F. A. and Esposito, F. (2008). Learning SHIQ+log rules for ontology evolution. In *Proceedings of the 5th Workshop on Semantic Web Applications and Perspectives (SWAP)*, volume 426 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- [Lisi and Malerba, 2003] Lisi, F. A. and Malerba, D. (2003). Ideal refinement of descriptions in AL-log. In Horváth, T., editor, *Inductive Logic Programming: 13th International Conference, ILP 2003, Szeged, Hungary, September 29-October 1, 2003, Proceedings*, volume 2835 of *Lecture Notes in Computer Science*, pages 215–232. Springer.

- [Lukasiewicz, 2008] Lukasiewicz, T. (2008). Expressive probabilistic description logics. *Artif. Intell.*, 172(6-7):852–883.
- [Michalski, 1980] Michalski, R. S. (1980). Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4):349–361.
- [Mitchell, 1997] Mitchell, T. (1997). *Machine Learning*. McGraw Hill, New York.
- [Mooney, 1997] Mooney, R. J. (1997). Inductive logic programming for natural language processing. *Inductive Logic Programming*, 1314:3–22. Times Cited: 1 Lecture Notes in Artificial Intelligence Article English Cited References Count: 63 Bn72g.
- [Mooney and Califf, 1995] Mooney, R. J. and Califf, M. E. (1995). Induction of first-order decision lists: Results on learning the past tense of english verbs. *J. Artif. Intell. Res. (JAIR)*, 3:1–24.
- [Muggleton, 1991] Muggleton (1991). Inductive logic programming. *NEWGEN: New Generation Computing*, 8.
- [Muggleton, 1992] Muggleton, S. (1992). *Inductive Logic Programming*. Academic Press, New York.
- [Muggleton, 1995] Muggleton, S. (1995). Inverse entailment and prolog. *New Generation Computing*, 13(3&4):245–286.
- [Muggleton, 1996a] Muggleton, S. (1996a). Learning from positive data. In Muggleton, S., editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 358–376. Springer-Verlag.
- [Muggleton, 1996b] Muggleton, S. (1996b). Learning from positive data. In Muggleton, S., editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 358–376. Springer-Verlag.
- [Muggleton and Feng, 1990] Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. In *ALT*, pages 368–381.
- [Muggleton and Raedt, 1994] Muggleton, S. and Raedt, L. D. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679.
- [Naumann et al., 2006] Naumann, F., Bilke, A., Bleiholder, J., and Weis, M. (2006). Data fusion in three steps: Resolving schema, tuple, and value inconsistencies. *IEEE Data Engineering Bulletin*, 29(2):21–31.

- [Nienhuys-Cheng and de Wolf, 1997] Nienhuys-Cheng, S.-H. and de Wolf, R., editors (1997). *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science*. Springer.
- [Nienhuys-Cheng et al., 1999] Nienhuys-Cheng, S.-H., Laer, W. V., Ramon, J., and Raedt, L. D. (1999). Generalizing refinement operators to learn prenex conjunctive normal forms. In Džeroski, S. and Flach, P., editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 245–256. Springer-Verlag.
- [Nienhuys-Cheng et al., 1993] Nienhuys-Cheng, S.-H., van der Laag, P. R. J., and van der Torre, L. W. N. (1993). Constructing refinement operators by decomposing logical implication. In Torasso, P., editor, *Advances in Artificial Intelligence: Proceedings of the 3rd Congress of the Italian Association for Artificial Intelligence (AI*IA '93)*, volume 728 of *LNAI*, pages 178–189, Torino, Italy. Springer.
- [Patel-Schneider and Horrocks, 2007] Patel-Schneider, P. F. and Horrocks, I. (2007). A comparison of two modelling paradigms in the Semantic Web. *Journal on Web Semantics*, 5(4):240–250.
- [Plotkin, 1971] Plotkin, G. D. (1971). *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University.
- [Quinlan and Cameron-Jones, 1993] Quinlan, J. R. and Cameron-Jones, R. M. (1993). FOIL: a midterm report. In Brazdil, P. B., editor, *European Conference on Machine Learning Proceedings (ECML-93)*, pages 3–20, Vienna. Springer-Verlag.
- [Raedt, 1996] Raedt, L. D., editor (1996). *Advances in Inductive Logic Programming*. IOS Press.
- [Raedt, 2005] Raedt, L. D. (2005). Statistical relational learning: An inductive logic programming perspective. In Jorge, A., Torgo, L., Brazdil, P., Camacho, R., and Gama, J., editors, *Knowledge Discovery in Databases: PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Porto, Portugal, October 3-7, 2005, Proceedings*, volume 3721 of *Lecture Notes in Computer Science*, pages 3–5. Springer.
- [Raedt et al., 2008] Raedt, L. D., Frasconi, P., Kersting, K., and Muggleton, S., editors (2008). *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*. Springer.
- [Ramanan, 2002] Ramanan, P. (2002). Efficient algorithms for minimizing tree pattern queries. In *SIGMOD '02: Proc. of the 2002 ACM SIGMOD Int. Conf. on Management of data*, pages 299–309. ACM.

- [Rector and Brandt, 2008] Rector, A. L. and Brandt, S. (2008). Why do it the hard way? The case for an expressive description logic for SNOMED. *J Am Med Inform Assoc*.
- [Reka and Albert-Laszlo, 2002] Reka, A. and Albert-Laszlo, B. (2002). Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97.
- [Richards and Mooney, 1995] Richards, B. L. and Mooney, R. J. (1995). Refinement of first-order Horn-clause domain theories. *Machine Learning*, 19(2):95–131.
- [Riechert et al., 2007a] Riechert, T., Lauenroth, K., and Lehmann, J. (2007a). Semantisch unterstütztes requirements engineering. In *Proceedings of the SABRE-07 SoftWiki Workshop*.
- [Riechert et al., 2007b] Riechert, T., Lauenroth, K., Lehmann, J., and Auer, S. (2007b). Towards semantic based requirements engineering. In *Proceedings of the 7th International Conference on Knowledge Management (I-KNOW)*.
- [Rudolph, 2004] Rudolph, S. (2004). Exploring relational structures via FLE. In Wolff, K. E., Pfeiffer, H. D., and Delugach, H. S., editors, *Conceptual Structures at Work: 12th International Conference on Conceptual Structures, ICCS 2004, Huntsville, AL, USA, July 19-23, 2004. Proceedings*, volume 3127 of *Lecture Notes in Computer Science*, pages 196–212. Springer.
- [Schwitter et al., 2008] Schwitter, R., Kaljurand, K., Cregan, A., Dolbear, C., and Hart, G. (2008). A comparison of three controlled natural languages for OWL 1.1. In Clark, K. and Patel-Schneider, P. F., editors, *Proceedings of the Fourth International Workshop OWL: Experiences and Directions, OWLED2008DC, Washington, D.C., April 2008*. Available from <http://www.webont.org/owled/2008dc>.
- [Seidenberg and Rector, 2006] Seidenberg, J. and Rector, A. L. (2006). Web ontology segmentation: analysis, classification and use. In Carr, L., Roure, D. D., Iyengar, A., Goble, C. A., and Dahlin, M., editors, *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006*, pages 13–22. ACM.
- [Shapiro, 1991] Shapiro, E. Y. (1991). Inductive inference of theories from facts. In Lassez, J. L. and Plotkin, G. D., editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 199–255. The MIT Press.
- [Srinivasan et al., 1997] Srinivasan, A., King, R. D., Muggleton, S., and Sternberg, M. J. E. (1997). Carcinogenesis predictions using ILP. In Džeroski, S. and Lavrač, N., editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 273–287. Springer-Verlag.

- [Staab and Studer, 2004] Staab, S. and Studer, R., editors (2004). *Handbook on Ontologies*. International Handbooks on Information Systems. Springer Verlag, Heidelberg.
- [Straccia, 2001] Straccia, U. (2001). Reasoning within fuzzy description logics. *J. Artif. Intell. Res. (JAIR)*, 14:137–166.
- [Suchanek et al., 2007] Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: a core of semantic knowledge. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 697–706, New York, NY, USA. ACM Press.
- [Suchanek et al., 2008] Suchanek, F. M., Kasneci, G., and Weikum, G. (2008). Yago: A large ontology from wikipedia and wordnet. *Journal of Web Semantics*, 6(3):203–217.
- [Swartz, 2002] Swartz, A. (2002). Musicbrainz: A semantic web service. *IEEE Intelligent Systems*, 17(1):76–77.
- [T. Berners-Lee et al., 2006] T. Berners-Lee et al. (2006). Tabulator: Exploring and analyzing linked data on the semantic web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop*.
- [The Gene Ontology Consortium, 2000] The Gene Ontology Consortium (2000). Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29.
- [Toivonen et al., 2003] Toivonen, H., Srinivasan, A., King, R. D., Kramer, S., and Helma, C. (2003). Statistical evaluation of the predictive toxicology challenge 2000-2001. *Bioinformatics*, 19(10):1183–1193.
- [Tummarello et al., 2007] Tummarello, G., Delbru, R., and Oren, E. (2007). Sindice.com: Weaving the Open Linked Data. In *Proceedings of the 6th International Semantic Web Conference*.
- [van der Laag and Nienhuys-Cheng, 1994] van der Laag, P. R. J. and Nienhuys-Cheng, S.-H. (1994). Existence and nonexistence of complete refinement operators. In Bergadano, F. and Raedt, L. D., editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 307–322. Springer-Verlag.
- [Völker et al., 2007a] Völker, J., Hitzler, P., and Cimiano, P. (2007a). Acquisition of OWL DL axioms from lexical resources. In *Proceedings of the 4th European Semantic Web Conference (ESWC)*, volume 4519 of *LNCS*, pages 670–685. Springer.
- [Völker and Rudolph, 2008] Völker, J. and Rudolph, S. (2008). Fostering web intelligence by semi-automatic OWL ontology refinement. In *Web Intelligence*, pages 454–460. IEEE.

- [Völker et al., 2007b] Völker, J., Vrandečić, D., Sure, Y., and Hotho, A. (2007b). Learning disjointness. In *Proceedings of the 4th European Semantic Web Conference (ESWC)*, volume 4519 of *LNCS*, pages 175–189. Springer.
- [Šebelík and Štěpánek, 1982] Šebelík, J. and Štěpánek, P. (1982). Horn clause programs for recursive functions. In Clark, K. and Tärnlund, S.-Å., editors, *Logic programming*, pages 324–340. Academic Press, New York.
- [Železný et al., 2003] Železný, F., Srinivasan, A., and Page, D. (2003). Lattice-search runtime distributions may be heavy-tailed. In Matwin, S. and Sammut, C., editors, *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 333–345. Springer-Verlag.
- [Watanabe and Muggleton, 2009] Watanabe, H. and Muggleton, S. (2009). Can ILP be applied to large dataset? In *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium*.
- [Ziegler et al., 2008] Ziegler, C.-N., Lausen, G., and Konstan, J. A. (2008). On exploiting classification taxonomies in recommender systems. *AI Commun*, 21(2-3):97–125.

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, den 5.1.2010

Jens Lehmann