Web Ontology Reasoning with Logic Databases

Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften
(Dr. rer. pol.)
von der Fakultät für
Wirtschaftswissenschaften
der Universität Fridericiana zu Karlsruhe
genehmigte
DISSERTATION
von
Dipl.-Inform. Raphael Volz

Tag der mündlichen Prüfung: 17. Februar 2004
Referent: Prof. Dr. Rudi Studer
1. Korreferent: Prof. Dr. Waldmann
2. Korreferentin: Prof. Carole Goble, University of Manchester
2004 Karlsruhe

To my family.

Abstract

This dissertation addresses the problem of reasoning with Web ontologies in logic databases. The main contribution is the enabling of Web ontology reasoning with logic databases for the Web ontology language (OWL), an international standard that has been recently proposed by the World Wide Web Consortium (W3C).

On the theoretical side, we show which fragment of OWL can be correctly supported by logic databases. On the practical side, we show that logic databases allow to overcome the predominant limitation of current approaches to ABox reasoning with Description Logics (DL) such as OWL.

Our work specifically addresses the following questions:

- *Which fragment of OWL can be implemented in logic databases ?* We invent a strategy to identify the fragment, which is based on the translation of DL axioms into first-order logic, followed by a transformation into a conjunction of Horn formulae. These formulae can be translated into the different Logic Programming languages which are supported by logic databases.

- *What are the theoretical properties of the fragment ?* We define Description Logic Programs (DLP) as a family of new ontology languages $\mathcal{L}_i$, which precisely contains the fragment. We utilize DLP to study the theoretical properties of the fragment. In particular, we show that we can solve ABox related reasoning problems, independent of the TBox, with a low polynomial complexity $O(|\mathcal{A}^{DLP}_{i\leq 2}|)^4$ in the size of the DLP ABox $\mathcal{A}^{DLP}$.

- *How can DL reasoning be reduced to operations on logic databases ?* We show which and how DL reasoning problems can be reduced to operations on logic databases and identify the limitations of the reduction.

- *How useful is the related fragment as a practical Web ontology language ?* We answer this question by showing two separate hypotheses.

  Hypothesis 1: The fragment is sufficient to express most available Web ontologies. We analyze the largest currently available collection of Web ontologies and check which fragment of those ontologies can be expressed in DLP. We quantify *most* in two respects. Firstly, we show that the different DLP languages

suffice to express 77 % ($\mathcal{L}_0$) – 87 % ($\mathcal{L}_3$) of the analyzed ontologies. Secondly, we could express 93% ($\mathcal{L}_0$) – 99 % (($\mathcal{L}_3$)) of the individual axioms in the analyzed ontologies.

Hypothesis 2: We can efficiently and scalably reason with DLP. We will show that logic databases not only exhibit satisfactory performance for ABox-related reasoning problems in DLP knowledge bases but also compare favorably with the only state of the art DL reasoner which supports ABox-reasoning.

- *How can we improve the performance of ABox-related reasoning problems ?* Certain language features of OWL (such as equality) deteriorate the performance of ABox-reasoning severely. We propose the materialization of knowledge bases to store computed entailments and speed up reasoning. We show how we can maintain a materialization incrementally in the case of changes.

Put together, the results developed in this thesis allow scalable and practical solutions to ABox-related reasoning problems for OWL-based applications. Our results are based on delegating ABox reasoning problems to logic databases. The thesis covers all necessary stages to obtain this delegation, starting from the identification of the (sub)languages for which this delegation can be achieved, via the theoretical analysis of the properties of these languages, to the assessment of the performance obtained by the delegation.

Even though logic databases have been used for Semantic Web applications from the early days of the Semantic Web, no comprehensive solution, which guarantees the correctness of the obtained answers, has been presented before for OWL or its predecessors[1].

[1] DAML+OIL, DAML-ONT and OIL.

Contents

Abstract v

I. Foundations 1

1. Introduction 3
1.1. Context . . . 3
1.1.1. Machine Readability . . . 5
1.1.2. Machine Interpretability . . . 5
1.2. Motivation . . . 7
1.2.1. Semantic Web Architecture . . . 7
1.2.2. Current Limitations of OWL . . . 8
1.3. Contribution . . . 9
1.4. Reader's Guide . . . 11

2. Logical Foundations 13
2.1. Introduction . . . 13
2.2. First-Order Logic (FOL) . . . 14
2.2.1. Syntax . . . 14
2.2.2. Semantics . . . 16
2.2.3. Normal Forms . . . 18
2.3. Logic Programming . . . 20
2.3.1. Syntax . . . 20
2.3.2. Semantics . . . 21
2.3.3. Variants and Extensions . . . 23
2.3.4. Reasoning Problems . . . 27
2.3.5. Complexity . . . 27
2.3.6. Evaluation Strategies . . . 28
2.4. Description Logics . . . 31
2.4.1. Syntax . . . 31
2.4.2. Semantics . . . 34
2.4.3. Reasoning Problems . . . 38

2.4.4. Complexity . . . 40
2.4.5. Evaluation Strategies . . . 41

3. The Semantic Web 43
3.1. Introduction . . . 44
3.2. Syntax Layer . . . 46
3.2.1. Extensible Markup Language (XML) . . . 46
3.2.2. XML Namespaces . . . 47
3.2.3. XML Schema Languages . . . 47
3.2.4. Discussion . . . 48
3.3. Data Layer . . . 49
3.3.1. Resource Description Framework (RDF) . . . 49
3.3.2. Data Model . . . 50
3.3.3. RDF Syntax . . . 51
3.3.4. RDF Schema (RDFS) . . . 53
3.3.5. RDF Semantics . . . 55
3.3.6. Axiomatic Semantics . . . 58
3.4. Ontology Layer . . . 60
3.4.1. Web Ontology Language (OWL) . . . 60
3.4.2. Syntax . . . 61
3.4.3. Semantics . . . 64
3.4.4. Complexity of OWL . . . 69
3.4.5. Discussion . . . 69
3.5. The Logic Layer - Rules . . . 70
3.5.1. XML-based Approach - RuleML . . . 70
3.5.2. RDF-based Approaches - N3 and Triple . . . 72
3.5.3. OWL-based Approaches . . . 74

II. Description Logic Programs 75

4. The Relation between Description Logics and Logic Programming 79
4.1. Introduction . . . 80
4.1.1. Motivation . . . 80
4.1.2. Related Work . . . 80
4.1.3. Assumptions . . . 81
4.2. Approach . . . 82
4.2.1. Semantic Correspondence . . . 82
4.2.2. Translation . . . 82
4.2.3. Logic Programming variants . . . 83
4.3. Preprocessing Techniques . . . 83

4.3.1. Normalization/Simplification of class descriptions 84
4.3.2. Normalization of axioms 86
4.3.3. Structural Transformation 87
4.4. From DL to FOL to LP 87
4.4.1. From DL to FOL 88
4.4.2. From FOL to LP 92
4.5. Finding the Intersection 93
4.5.1. Assertions 94
4.5.2. Atomic Classes and Properties 95
4.5.3. Universal Class and Empty Class 95
4.5.4. Boolean Class Descriptions 97
4.5.5. Enumeration 101
4.5.6. Property Restrictions 103
4.5.7. Property Characteristics 109
4.5.8. Summary 110
4.6. Conclusion 110
4.6.1. Contribution 110
4.6.2. Further uses 112

5. Description Logic Programs 113
5.1. Syntax 114
5.1.1. Datalog 114
5.1.2. Datalog with Equality 116
5.1.3. Datalog with Equality and Integrity Constraints 117
5.1.4. Prolog 118
5.1.5. Exchanging DLP Ontologies 119
5.2. Semantics 119
5.2.1. Translation of Classes 119
5.2.2. Translation of Properties and Assertions 121
5.2.3. Postprocessing 122
5.2.4. Translation Semantics 123
5.2.5. Terminological Cycles 123
5.3. Expressivity 125
5.3.1. Expressivity in Theory 126
5.3.2. Expressivity in Practise 128
5.4. Reasoning Problems 130
5.4.1. Reducing DL Queries to LP Queries 130
5.4.2. Class-related ABox Problems 130
5.4.3. Property-related ABox Problems 134
5.4.4. TBox Problems 135
5.4.5. Descriptive Information 140

5.5. Complexity . . . 141
5.5.1. Datalog-variants of DLP . . . 141
5.5.2. Prolog-variant of DLP . . . 143
5.5.3. LP Translation . . . 143
5.6. Conclusion . . . 143

6. Materialization **145**
6.1. Introduction . . . 146
6.1.1. Motivation . . . 146
6.1.2. Related Work . . . 147
6.2. Maintenance Principles . . . 149
6.2.1. Updates to Facts . . . 149
6.2.2. Updates to Rules . . . 151
6.2.3. Differentiating Between Asserted and Entailed Information . 152
6.3. Maintaining Changing Facts . . . 152
6.3.1. Approach . . . 153
6.3.2. Maintenance Rewritings . . . 154
6.3.3. Evaluating Maintenance Programs . . . 158
6.4. Maintaining Changing Rules . . . 159
6.4.1. Approach . . . 159
6.4.2. Maintenance Rewriting . . . 160
6.4.3. Evaluating Maintenance Programs . . . 161
6.5. Conclusion . . . 165
6.5.1. Contribution . . . 165
6.5.2. Further Uses . . . 166
6.5.3. Materializing RDF Rules . . . 167

III. Finale **171**

7. Implementation **173**
7.1. Introduction . . . 173
7.2. KAON . . . 175
7.2.1. Client Applications . . . 176
7.2.2. Middleware Layer . . . 176
7.2.3. External Services Layer . . . 177
7.3. OWL API . . . 178
7.3.1. Aspects of Functionality . . . 178
7.3.2. Modelling . . . 179
7.3.3. Inferencing . . . 180
7.4. DLP Prototype . . . 180

7.4.1. Prototype Architecture . . . 180
7.4.2. Compilation . . . 182
7.4.3. Translation and Transformation . . . 183
7.4.4. Variants of Logic Databases . . . 185
7.5. Materialization . . . 187
7.6. Related Work . . . 188
7.6.1. Infrastructures for ontology-based applications . . . 188
7.6.2. Ontology and Knowledge Base APIs . . . 189
7.7. Conclusion . . . 190

8. Performance Analysis 193
8.1. Introduction . . . 194
8.2. Analyis Environment . . . 196
8.2.1. Selection of Systems . . . 196
8.2.2. System Descriptions . . . 196
8.2.3. Testing Methodology . . . 198
8.3. Ontology Library Analysis . . . 198
8.3.1. Library Content . . . 199
8.3.2. Average Characterizations . . . 200
8.3.3. Distributions in Class Definitions . . . 202
8.3.4. Discussion . . . 205
8.4. Comparative Performance Analysis . . . 206
8.4.1. Goals and Assumptions . . . 206
8.4.2. Knowledge Bases . . . 206
8.4.3. Results . . . 208
8.4.4. Discussion . . . 213
8.5. Expressivity Performance Analysis . . . 213
8.5.1. Goals and Assumptions . . . 214
8.5.2. Knowledge Bases with Equality . . . 215
8.5.3. Knowledge Bases with Skolem Functions . . . 218
8.5.4. Discussion . . . 219
8.6. Incremental Maintenance Analysis . . . 219
8.6.1. Goals and Assumptions . . . 220
8.6.2. Knowledge Bases . . . 220
8.6.3. Results . . . 221
8.6.4. Discussion . . . 221
8.7. Conclusion . . . 222

9. Conclusion **225**
9.1. Summary . . . 225
9.2. Further Applications . . . 228
9.3. An Appeal . . . 229

IV. Appendix **231**

A. BNF Grammar for DLP **233**

B. DLP Evaluation Results **237**
B.1. Comparative Performance Results . . . 237
B.1.1. Instance Retrieval . . . 237
B.1.2. Property Filler Retrieval . . . 239
B.1.3. Class Subsumption . . . 240
B.2. Expressivity Performance Results . . . 241
B.3. Materialization . . . 243

C. DAML Ontology Library in DLP **245**
C.1. Individual Evaluation Results . . . 245
C.1.1. Converted DAML+OIL Ontologies . . . 247
C.1.2. Excluded Ontologies . . . 249

D. Statistical Analysis of DAML+OIL ontologies **253**
D.1. Counts . . . 255
D.1.1. Absolute Counts . . . 255
D.1.2. Relative Counts . . . 256
D.2. Distributions . . . 258
D.2.1. Super-Class Distributions . . . 258
D.2.2. Restriction Distributions . . . 259

Bibliography **261**

List of Figures

1.1. Christmas Oratorio Knowledge Base 4

3.1. Semantic Web Layer Cake 45
3.2. Hierarchy of RDF-compatible XML Schema datatypes 49
3.3. RDFS Property Hierarchy 53
3.4. Class Hierarchy for RDF Schema 54
3.5. RDF Model Theory: Interpretation 55

4.1. Expressive Overlap of DL with LP. 88

5.1. DLP Language Variants 115

6.1. OntoWeb Architecture 147
6.2. Bach Family Tree Excerpt 150
6.3. Information Flow in Maintenance Programs: (a) Maintenance for Facts; (b) Maintenance for Rules 161

7.1. KAON Architecture 175
7.2. DLP Prototype 181

8.1. Distributions in Class Descriptions 203
8.2. Instance Retrieval on PO KBs 208
8.3. Ratio of Response Time to Number of Individuals 209
8.4. Response Time for Property Retrieval Tests 211
8.5. KB structure 215
8.6. Response Time of KAON with Equality 217
8.7. Cost of Maintenance Procedures for 10% Change in Facts 222

List of Tables

2.1. Terminology for Different Types of Clauses 20
2.2. DL Class and Property Constructors 32
2.3. DL Axioms 33
2.4. Extension of a DL with a Concrete Domain 34
2.5. The Bach Family Knowledge Base 35
2.6. Complexity of Satisfiability for $\mathcal{AL}$ languages w/o TBox 40
2.7. Complexity of Satisfiability for $\mathcal{S}$ languages 41

3.1. Datalog Axiomatization of RDF(S) Constructors 59
3.2. Syntactic Shortcuts 64
3.3. OWL DL Class Constructors (Abbr.) 65
3.4. OWL DL Axioms 67
3.5. OWL DL Assertions 68
3.6. Property Chaining in Rule Languages 71

4.1. Equivalences for Normalization/Simplification of Class Descriptions 84
4.2. Normalization and Simplification Rules 85
4.3. FOL Translation of $\mathcal{SHINO}$ Class Descriptions 90
4.4. FOL Translation of $\mathcal{SHINO}$ Property Axioms 90
4.5. FOL Translation of ABox Assertions 91
4.6. Translation of Property Characteristics to LP 110
4.7. Expressivity of Different LP Variants 111

5.1. DLP TBox axioms 116
5.2. Translation of Class Constructors to Logic Programs 120
5.3. Translation of Properties and Assertions to Logic Programs 122
5.4. Additional DLP Convenience Primitives 126
5.5. Translation of ODMG ODL Class Specification to DLP Axioms 127
5.6. DAML.ORG Ontologies in DLP 129

6.1. Example Knowledge Base 151
6.2. Rewriting Functions 156

6.3. Rewriting Functions 160
6.4. Datalog Translation of RDF Rule Languages 167

7.1. Translation with Integrated Transformation 184

8.1. Average Usage of Class Constructors Across Ontologies 200
8.2. Average Usage of Property Definitions 201
8.3. Ratio between Selected Language Primitives (Across all Ontologies) 202
8.4. TBox Variants 206
8.5. ABox Variants 207
8.6. Property Variants 207
8.7. Input Knowledge Base Size 208
8.8. Response Time to Class Subsumption Problems 212
8.9. Size and Content of Equality TBox 216
8.10. Response time of KAON with Equality 217
8.11. Size and Content of TBox with Existential Restrictions 218
8.12. Size of Maintenance 221
8.13. Costs of Materialization Procedures (in ms) 221

Part I.

Foundations

1. Introduction

Fromme Musen, meine Glieder,
Singt nicht längst bekannte Lieder,
Dieser Tag sei eure Lust,
Füllt mit Freuden eure Brust,
Werft so Kiel als Schriften nieder,
Und erfreut euch dreimal wieder.
5th Aria (Alto) of Cantata
"Tönet, ihr Pauken, erschallet, Trompeten" (BWV 214)
Johann-Sebastian Bach
`http://www.cs.ualberta.ca/~wfb/cantatas/214.html`

1.1. Context

The distributed information systems of today, such as the World Wide Web (WWW), peer to peer (P2P) file sharing systems and company intranets, resemble a poorly mapped geography. Our insights into available information in those systems are based on keyword searches, sometimes abetted by clever use of document connectivity (Page et al., 1998) and usage patterns. Consider the following question:

Example 1.1.1 *Find the lyrics of those Birthday Cantatas that were reused in Johann-Sebastian Bachs' Christmas Oratorio.*

Today, answering this question would require a (non-deterministic) sequence of consecutive keyword searches.[1] The non-determinism of the sequence is due to the fact, that each answer must be interpreted by humans to formulate the next search, which may (or may not) eventually lead to the intended results.

It would be difficult to construct a software system that would be capable of answering the above and similar questions, in a general manner, on any network

[1] **Quiz question:** Which web pages provide the lyrics of those cantatas ?

topology (be it Web or Peer-To-Peer) and leading to results that answer the question. The main difficulty lies in the fact that the computation itself must necessarily go beyond keywords and requires an interpretation of additional data, i.e. metadata, that describe the content of network-accessible resources. For example, the relevant aspects of the natural language content of a document must be described through metadata (cf. Figure 1.1, where BWV 213, 214 and 248 are identifiers for the metadata elements that describe some compositions of Johann-Sebastian Bach).

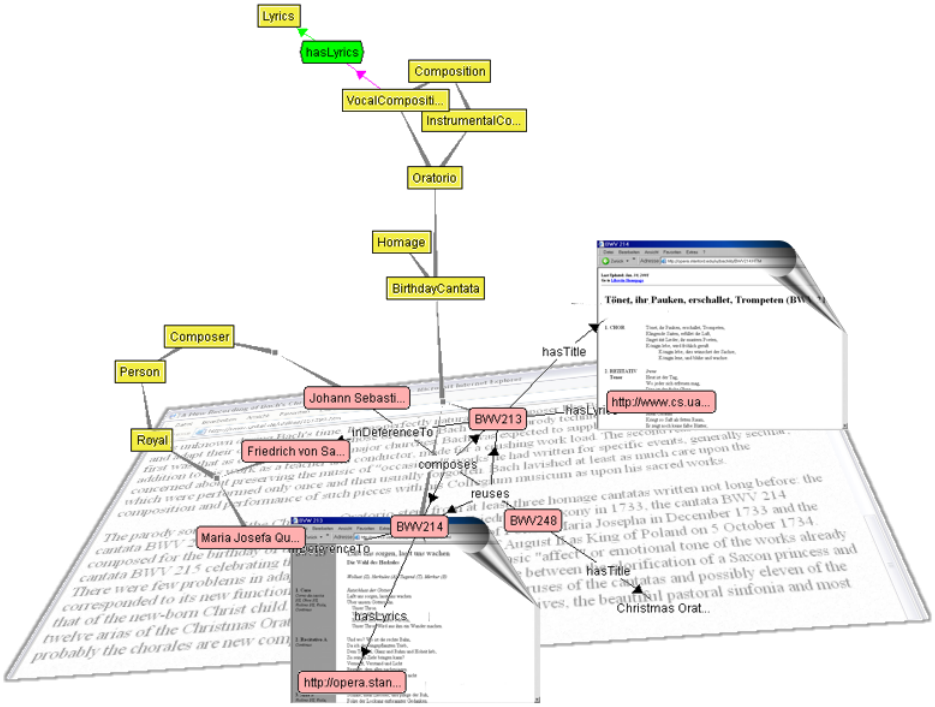


Figure 1.1.: Christmas Oratorio Knowledge Base

In the last five years, the Semantic Web standardization activity of the World Wide Web Consortium (W3C) was focused to design the basic technical standards that allow the representation of metadata on the Web, providing technologies that eventually enable applications to derive answers on questions like Example 1.1.1. In parallel, these standards can be seen as cornerstones for the development of future

large-scale Semantic Web applications. The standards have primarily focused on two main aspects:

1. machine readability
2. machine interpretability

1.1.1. Machine Readability

In a nutshell, this aspect is covered by the definition of a framework for the description of resources in distributed systems, i.e. the Resource Description Framework (RDF) (Lassila & Swick, 1999). RDF can be understood as a semi-structured data model for distributed, *relational metadata*.

The collection of metadata that describes the information sources using the RDF data model is the central part of a Semantic Web application. We will call this collection *knowledge base*[2]. As mentioned above, the knowledge base is expected to be used by information consumers to locate information sources through queries to the metadata.

Clearly, information in Semantic Web applications is characterized by the very fact that information is distributed. Therefore, a knowledge base typically contains metadata from a collection of information sources, e.g. several Web pages. For the context of this thesis we assume the currently dominant approach of aggregating distributed metadata into a single knowledge base.

1.1.2. Machine Interpretability

To answer the question of Example 1.1.1 on a knowledge base like Figure 1.1, information consumers and providers need to agree on the denotation of a *common vocabulary*, i.e. proper nouns[3] such as "Christmas Oratorio", and "Johann-Sebastian Bach", concepts[4] such as "birthday" and "cantata" as well as the relation[5] "reuses", to make queries possible.

[2]The knowledge base must not necessarily be expressed in the RDF data model. This is, however, the currently dominant approach in the Semantic Web.

[3]The red nodes in Figure 1.1. Such nodes are called *Individuals* in the terminology of Web ontology languages.

[4]The yellow nodes in Figure 1.1. Such nodes are called *Classes* in the terminology of Web ontology languages.

[5]The green nodes in Figure 1.1. Such nodes are called *Properties* in the terminology of Web ontology languages.

Web ontology languages allow to describe this vocabulary by means of logical axioms. We will differentiate between two basic categories of axioms, where the first category *(TBox)* contains the description of the *terminology* in a domain, and the second part *(ABox)* contains the actual *assertions* that make use of the terminology, i.e. provide the data that users are usually interested in. Two Web ontology languages, the RDF vocabulary description language (RDFS) (Brickley & Guha, 2003) and the Web Ontology Language (OWL) (McGuinness & van Harmelen, 2003), have been standardized recently and play a central role in this thesis. The languages mainly differ in expressivity. However, both languages are grounded on the assignment of formal semantics to their language constructs, which allow machines to interpret information such that new *implicit* facts can be derived from asserted facts using a form of symbolic processing, i.e. *logical entailment*.

This logical entailment of implicit information is mainly based on the interpretation of *axioms* in the TBox, which relate terminological *descriptions* with each other. For example, in the knowledge base of Figure 1.1 a machine can derive that every BIRTHDAYCANTATA such as BWV214 is also a COMPOSITION through the interpretation of subsumption axioms stated on the classes of the TBox.

In the brief history of the Semantic Web, most applications, e.g. (Decker et al., 1998), have implemented this logical entailment either directly using Logic Programming techniques, e.g. (Berners-Lee, 2000a; Roo, 2002), or by relying on (available) logic databases (Motik et al., 2002a; Sintek & Decker, 2001). This strategy, however, is no longer directly possible with the advent of the recently standardized Web Ontology Language (OWL), which relies on a logical paradigm called Description Logics (Baader et al., 2003).

We will study how we can continue to use logic databases[6] in the context of OWL. *Logic databases*, in our sense, are declarative, rule-based systems that apply Logic Programming techniques for expressing deductions and logical queries concerning the content of a database. We therefore investigate the relationship between Description Logics and the different variants of those Logic Programming languages, which are frequently used in logic databases. This analysis and the study of its practical consequences for Semantic Web applications is the fundamental contribution of this thesis.

In the following we will use the terms Logic Programming and logic databases interchangeably, using whatever form is most convenient in the context.

[6] We use the term logic database over the older term deductive databases since the later is very closely associated with Datalog, a particular Logic Programming language that is frequently used in logic databases. Modern logic databases such as XSB (Sagonas et al., 1994) and CORAL (Ramakrishnan et al., 1994) support more expressive Logic Programming languages that include function symbols and nested expressions. Furthermore, several lectures, e.g. `http://user.it.uu.se/~voronkorov/ddb.htm` nowadays use this term.

1.2. Motivation

This section motivates why the Logic Programming techniques that are applied in logic databases are useful in the context of the Semantic Web. We can differentiate two main motivations. Firstly, Logic Programming can provide suitable functionality for the general Semantic Web architecture. Secondly, Logic Programming allows to bypass current limitations of OWL.

1.2.1. Semantic Web Architecture

The usage of Logic Programming techniques for the purpose of the Semantic Web is considered to be central (Berners-Lee, 1998) for the future development of the Semantic Web, since - from the very beginning - the vision of the Semantic Web (Berners-Lee, 1999) includes, specifically, rules as well as ontologies.

Rule Language A key requirement for the Semantic Web's architecture overall, then, is to be able to express rules on ontology-based knowledge bases, viz. to create and reason with rule-bases that mention vocabulary specified by ontologies, and to do so in a semantically coherent manner. For OWL, this layering must reduce the expressiveness of the language, since a direct and naïve extension would immediately yield an undecidable language (Levy & Rousset, 1996; Schmidt-Schauß, 1989) for which only highly intractable first-order theorem provers can be used as a basis for logical entailment.

Query Language At the time of writing this thesis, the W3C gathers a new group for the standardization of a Semantic Web query language [7]. The Description Logic which underlies OWL can be used as a query language, but its capabilities with respect to querying are rather low, since only conjunctive queries — the least expressive query language usually considered in database research — can be supported (Horrocks & Tessaris, 2002).

Expressive query languages are the main form of interaction with logic databases, which offer extensive facilities for this purpose. Moreover, the theoretical and practical properties of the logic database approach are well understood and have been studied extensively in the last three decades. Hence, we can expect useful synergies by combining Web Ontology languages with the logic database paradigm. These synergies allow to state expressive queries with respect to ontology-based data.

[7] Cf. `http://www.w3.org/2002/11/swv2/Activity#proposal`.

1.2.2. Current Limitations of OWL

Since OWL is basically a notational variant of a Description Logic (Horrocks & Patel-Schneider, 2003), it shares its benefits and limitations with those logics.

Tractability One of the main limitations of Description Logics (such as OWL) are their intractability (Nebel, 1990) and lack of algorithms which allow to reason with non-trivial ABoxes in practise. Restricted variants of Logic Programming languages such as Datalog, i.e. the language which is supported by most Logic Databases, however, are tractable and optimized reasoning algorithms for reasoning with large ABoxes, such as the Magic Sets technique (Ramakrishnan, 1991) or Tabling (Swift & Warren, 1994), are well known.

Bootstrapping the Semantic Web Until now, Description Logics have been developed by a small research community. Therefore, they have not yet received the same public attention as other approaches to information modeling , such as the Entity-Relationship (Chen, 1976) or object-oriented models (Cattell et al., 2000; Object Management Group, 2003). In particular, they have not - yet - found its way into main computer science curricula. While this effect might be temporary, it is nevertheless useful to facilitate the transition by relating the elements of the language with the elements of other well-known modeling languages. This relation can also be used to bootstrap (Stojanovic et al., 2002) the Semantic Web by reusing available information models.

System base Similarly - and partly dependent on the previous limitation - only three systems, i.e. FaCT (Horrocks et al., 1999), Racer (Haarslev & Moller, 2001) and Cerebra (Network Inference Inc., 2003), are able to provide limited support for reasoning with OWL today. All three systems are incomplete and do not support all language features. For example, FaCT (Horrocks et al., 1999) offers no support at all for ABoxes, while Racer (Haarslev & Moller, 2001) departs from the OWL semantics by taking the Unique Names Assumption (UNA). The logic database world, however, can provide an abundance of both commercial and open-source systems, which can be applied for the purpose of Semantic Web applications, if the ontology language fits to the Logic Programming language supported by these systems.

Legacy data integration Today, the majority of data on the Web is no longer static but resides in databases, e.g. to dynamically generate web pages. Exposing this data to the Semantic Web will require the access to the content of underlying

databases (Volz et al., 2004). Description Logic reasoners such as FaCT (Horrocks et al., 1999) or Racer (Haarslev & Moller, 2001) currently require the replication of that data to enable ontology-based processing. Logic databases such as XSB (Sagonas et al., 1994), Ontobroker (Decker et al., 1999) or CORAL (Ramakrishnan et al., 1994), however, can access relational databases directly through built-in predicates. Hence, Logic Programming allows closer interaction with externally stored data, i.e. data can remain where it can be handled most efficiently - in relational databases. Furthermore, more restricted variants of Logic Programming languages can be directly implemented on top of SQL99-compliant relational databases (Ullman, 1988), which would increase the possible system basis further.

Ontology Transformation We can expect that the de-centralized nature of the Semantic Web makes it inevitable that communities will evolve that use disparate ontologies. The assumption that both information providers and information consumers share one ontology will then no longer be valid, i.e. leading to the situation that information can not be shared, viz. results, which would match the query (if the correct vocabulary were used) can not be found. We then need to mediate between distributed data using mappings between ontologies (Rousset, 2002). Different proposals (Maedche et al., 2002a; Omelayenko, 2002; Noy & Musen, 2000) have already been made to specify mappings between ontologies. Logic databases can provide the functionality needed to execute the necessary transformations specified by such mappings.

1.3. Contribution

We address the general goal of Web ontology reasoning with logic databases in context of the recently proposed Web Ontology Language (OWL), which can be understood as a notational variant of a Description Logic. In this context our goal implies the following research questions:

- *Which fragment of OWL can be implemented in logic databases ?* We outline a strategy to identify the respective fragment of OWL. Our strategy is based on the translation of knowledge bases into first-order logic, followed by a transformation into a conjunction of Horn formulae. The Horn formulae can then be syntactically translated into the different Logic Programming syntaxes that are supported by logic databases. We instantiate this strategy on OWL to identify the fragment of OWL that can be supported.
- *What are the theoretical properties of the fragment ?* We define Description Logic Programs (DLP) (Grosof et al., 2003) in as a family of new ontology languages,

which are constituted by those OWL language primitives that can be supported by logic databases. We utilize the DLP family to study the theoretical properties, i.e. complexity and expressivity, of the fragment.

- *How can DL reasoning be reduced to queries on logic databases ?* We show how the various DL reasoning problems can be reduced to queries on logic databases. We present the procedure that is necessary to obtain the reduction and identify the practical limitations of the reduction.

- *Is the fragment sufficient to express most available Web ontologies ?* We analyze the largest currently available collection of Web ontologies and check which fragment of those ontologies can be expressed in the DLP languages. We quantify *most* in two respects. Firstly, we study which ontologies can be expressed completely. Secondly, we study which percentage of all axioms in the ontology can be represented in DLP.

- *Can logic databases efficiently and scalably reason with the fragment ?* We study whether logic databases exhibit satisfactory performance in solving DL reasoning problems. To this extend, we compare the performance of logic databases with that of native Description Logic reasoners. We also evaluate how the usage of more expressive language constructors impact performance.

- *How can we improve the performance of ABox-related reasoning problems ?* As we will see in the performance analyses, the introduction of equality deteriorates the performance of logic databases severely. We therefore propose the materialization (Volz et al., 2003g; Volz et al., 2003f) of knowledge bases as a technique to store computed entailments, such as derived through an equality theory. We show how we can maintain the materialization of the knowledge base incrementally when the knowledge base is changed.

A novel family of ontology languages, which we call *Description Logic Programs (DLP)* (Grosof et al., 2003), is the basic tool used to answer the above reasoning questions. Our answers, however, show that the various DLP languages can be understood as sensible Web ontology languages on their own. In particular, DLP extends the existing Web ontology language RDF Schema (RDFS) (Brickley & Guha, 2003) with more expressive constructs, while maintaining

1. maximum *backwards compatibility* with available Logic Programming technology and

2. maximum *forward compatibility* with even more expressive languages like OWL.

Practical Implementations All techniques developed in this thesis have been implemented in prototype tools which are part of the KArlsruhe ONtology (KAON) tool suite (Bozsak et al., 2002), the genesis of which was largely influenced by the work of the author. The tool suite is practically applied in many industrial and research prototypes.

The following tools are a direct outcome of this thesis and can be downloaded at `http://kaon.semanticweb.org/`:

- *KAON DLP*[8] provides a reasoner for the DLP language,
- the incremental materialization algorithm is implemented[9] within the *KAON Datalog Engine*.

In parallel to the development of KAON, the OWL API[10] (Bechhofer et al., 2003b), a reusable component for arbitrary OWL applications has been developed[11] in context of the DAAD scholarship received by the author and the EU-funded research project WonderWeb.

1.4. Reader's Guide

The thesis is organized as follows. Chapter 2 on page 13 recapitulates well-known fundamentals of logics relevant to the thesis, in particular fundamentals of Logic Programming and logic databases as well as fundamentals of Description Logics. Chapter 3 on page 43 introduces standards currently relevant to the Semantic Web. It contains a discussion of the role of ontologies in this setting and shows how the diverse standards map to the logics presented in the previous chapter.

The second part of the thesis presents Description Logic Programs (DLP) - the ontology language upon which our work is based.

Chapter 4 analyzes how Description Logic and Logic Programming are related and identifies the fragment which can be translated from one language into the other. Chapter 5 on page 113 utilizes the identified fragment for the definition of the DLP family of ontology languages $\mathcal{L}_i$ and presents the semantics for these languages, which is based on a formal translation into Logic Programs. The chapter shows how Description Logic reasoning problems can be reduced to answering queries on

[8] Jointly implemented with Boris Motik.
[9] Jointly implemented with Boris Motik.
[10] `http://www.sourceforge.net/projects/owlapi`
[11] Jointly with Sean Bechhofer of the University of Manchester

Logic Programs and studies the expressiveness and the complexity of the new languages. In particular, we show the practical usefulness of the languages by demonstrating that most parts of currently available Web ontologies can be expressed in the new languages and that the family of languages is tractable. Chapter 6 on page 145 proposes the materialization of DLP ontologies and presents an approach for the incremental maintenance of materialized logic databases when changes to facts and rules occur.

The third part of the thesis presents the implementation and evaluation of DLP. Chapter 7 on page 173 presents the DLP prototype reasoner and KAON, the surrounding infrastructure, in which the DLP prototype is embedded. Chapter 8 on page 193 evaluates reasoning with the DLP languages in logic databases and shows that the performance of querying knowledge bases compares favorably with Description Logic engines. We also show that our approach to materialization improves the efficiency of query answering and that the materialization can be incrementally maintained with tolerable costs. Chapter 9 on page 225 summarizes the contributions of this thesis and outlines further applications.

2. Logical Foundations

Unanfechtbare Wahrheiten gibt es überhaupt nicht,
und wenn es welche gibt, so sind sie langweilig.
Theodor Fontane,
Der Stechlin
http://www.gutenberg2000.de/fontane/stechlin/stech01.htm

2.1. Introduction

This chapter provides well-known fundamentals of those logics which are relevant to this dissertation, viz. used in the subsequent chapters. In particular we briefly recapitulate the fundamentals of First-order Logic (Section 2.2), Logic Programming and Deductive Databases (Section 2.3 on page 20) and Description Logics (Section 2.4 on page 31). Readers that are familiar with either one of the logics might skip the respective sections.

We do not attempt to give a thorough introduction to those logics, but concisely capture the aspects that are relevant to this thesis. To this extent each section gives a precise syntactic and semantic definition of the logics. The sections on Logic Programming and Description Logics additionally state the different main reasoning problems given for the logics and discuss the computational complexity of those reasoning problems. We conclude our exposition of the latter two logics with a presentation of the main approaches to compute answers for the posed reasoning problems.

Readers interested in a more thorough treatment of the logics might refer to textbooks on First-order logic, e.g. (Schöning, 1995), Description Logics (Baader et al., 2003), and on Logic Programming, e.g. (Ceri et al., 1990) or (Lloyd, 1987). (Dantsin et al., 2001) surveys the various complexity and expressiveness results on different forms of Logic Programming, a good overview over the complexity results of various Description Logics is given in Chapter 3 of (Baader et al., 2003).

2.2. First-Order Logic (FOL)

First-order logic (FOL) is an often used logic and is more general than the other two families of logics presented in this chapter. Nearly all textbooks on logic, e.g. (Schöning, 1995), give an introduction to FOL, consequently we only briefly recapitulate the basic definitions of the syntax and semantics of FOL as far as needed in this dissertation.

2.2.1. Syntax

For defining the language of FOL, we need a vocabulary. This vocabulary is given by a signature.

Definition 2.2.1 (Signature) *A signature Σ is a 5-tuple (C, F, P, V, a) and consists of*

- *a set C of constant symbols*
- *a set F of function symbols*
- *a set P of predicate symbols*
- *a set V of variable symbols*
- *and a function $a : F \cup P \to \mathbb{N}$*

The function a states the arity of functions and predicates.

Constant symbols are special function symbols with arity 0, we consider them separately since one of the important characteristics of many subsets of FOL, e.g. Datalog or $\mathcal{L}^2$, is the absence of function symbols other than constants.

Definition 2.2.2 (Term) *The set of terms of a signature $\Sigma = (C, F, P, V, a)$ is defined by the smallest set T_Σ such that:*

1. *$c \in T_\Sigma$ for each constant symbol $c \in C$;*
2. *$v \in T_\Sigma$ for each variable $v \in V$;*
3. *if $f \in F$ with arity $a(f) = m$ and $t_1, \ldots, t_m \in T_\Sigma$ then $f(t_1, \ldots, t_m) \in T_\Sigma$.*

Definition 2.2.3 (Ground term) *A term $t \in T_\Sigma$ is called a ground term if t does not contain any variables.*

The definition of the set of terms might vary. For example, the definition for Datalog would not include function symbols.

Definition 2.2.4 (Atomic Formula) *The set of atomic formulas At_Σ over a signature Σ is defined as follows: $p(t_1, \ldots, t_n) \in At_\Sigma$, if $p \in P$, $a(p) = n$ and all $t_i \in T_\Sigma$ for $1 \leq i \leq n$.*

Definition 2.2.5 (Ground Atom) *A predicate $p(t_1, \ldots, t_n) \in At_\Sigma$ with $a(p) = n$ is called ground atom if all terms t_i for $1 \leq i \leq n$ are ground terms.*

Definition 2.2.6 (Formula) *The set For_Σ of formulas over a signature Σ is defined by the smallest set for which:*

- $At_\Sigma \subseteq For_\Sigma$;
- *if $\varphi \in For_\Sigma$ then $\neg\varphi \in For_\Sigma$;*
- *if $v \in V$ and $\varphi \in For_\Sigma$ then $(\forall v.\varphi) \in For_\Sigma$;*
- *if $\varphi \in For_\Sigma$ and $\psi \in For_\Sigma$ then $(\varphi \vee \psi) \in For_\Sigma$.*

Existential quantification and all other logical connectives such as conjunction ($\wedge$), implication ($\rightarrow$), etc. can be used in formulas using the well known equivalences. For example, the existential quantifier $(\exists v.\varphi)$ can be understood as shortcut for the formula $\neg(\forall v.\neg\varphi)$.

Definition 2.2.7 (Bound Variable) *A variable v is* bound *in the formula φ if φ contains an expression $(\forall v.\varphi)$.*

Definition 2.2.8 (Free Variable) *A variable v is* free *in a formula φ if φ contains the variable and the variable is not bound.*

Definition 2.2.9 (Sentence) *A sentence (or closed formula) is a formula that contains no free variables.*

Definition 2.2.10 (Literal) *An atomic formula $\varphi \in At_\Sigma$ is called a positive literal. $(\neg\varphi) \in For_\Sigma$ with $\varphi \in At_\Sigma$ is called a negative literal. A literal l is also called ground literal if φ is a ground atom. L_Σ refers to the set of all literals.*

Definition 2.2.11 (Closure of formula) *Let $V = \{v_1, \ldots, v_n\}$ be the set of free variables in a formula φ, we call the formula*

- *$\forall v_1 \ldots \forall v_n.\varphi$ universal closure of the formula φ, alternatively we write $\forall.\varphi$;*
- *$\exists v_1 \ldots \exists v_n.\varphi$ existential closure of the formula φ, alternatively we write $\exists.\varphi$.*

2.2.2. Semantics

One of the standard ways for giving semantics to FOL is by means of a model theory (Tarski, 1956).

Definition 2.2.12 (Interpretation) *An interpretation $\mathfrak{I} = (\Delta^{\mathfrak{I}}, \cdot^{\mathfrak{I}})$ consists of a non-empty set $\Delta^{\mathfrak{I}}$ (domain or universe) and an interpretation function $\cdot^{\mathfrak{I}}$, which assigns elements of a signature $\Sigma = (C, F, P, V, a)$ to $\Delta^{\mathfrak{I}}$ such that :*

- *$\cdot^{\mathfrak{I}}$ maps each constant $c \in C$ to an element $c_{\mathfrak{I}} \in \Delta^{\mathfrak{I}}$ which is the interpretation of c ($c^{\mathfrak{I}} = c_{\mathfrak{I}}$);*
- *$\cdot^{\mathfrak{I}}$ maps each function symbol $f \in F$ with $a(f) = n$ to a function*

  $$f_{\mathfrak{I}} : (\Delta^{\mathfrak{I}})^n \to \Delta^{\mathfrak{I}}$$

  which is the interpretation of f ($f^{\mathfrak{I}} = f_{\mathfrak{I}}$);
- *$\cdot^{\mathfrak{I}}$ maps each predicate symbol $p \in P$ with $a(p) = n$ to a function*

  $$p_{\mathfrak{I}} : (\Delta^{\mathfrak{I}})^n \to \{\mathit{false}, \mathit{true}\}$$

  which maps to a truth value ($p^{\mathfrak{I}} = p_{\mathfrak{I}}$).

Additionally we define a variable valuation function $\mathcal{V} : V \to \Delta^{\mathfrak{I}}$ which assigns an element $d \in \Delta^{\mathfrak{I}}$ to every variable $v \in V$.

The interpretation function $\mathfrak{I}$ gives meaning to constants, functions and predicates. The variable assignment function $\mathcal{V}$ gives meaning to variables. We now continue with defining the meaning of terms and formulas. To do so, a valuation (or denotation) connects a syntactic expression (terms and predicates) over a given signature with an interpretation of that signature.

Definition 2.2.13 (Term Denotation) *The denotation $[t]^{\mathfrak{I}}_{\mathcal{V}}$ of a term t in the interpretation $\mathfrak{I}$ under the variable assignment function $\mathcal{V}$ is defined as follows:*

- *$[v]^{\mathfrak{I}}_{\mathcal{V}} = \mathcal{V}(v)$, for all $v \in V$;*
- *$[c]^{\mathfrak{I}}_{\mathcal{V}} = c^{\mathfrak{I}}$, for all $c \in C$;*
- *$[f(t_1, \ldots, t_n)]^{\mathfrak{I}}_{\mathcal{V}} = f^{\mathfrak{I}}([t_1]^{\mathfrak{I}}_{\mathcal{V}}, \ldots, [t_n]^{\mathfrak{I}}_{\mathcal{V}})$, for all $f \in F, t_i \in T_{\Sigma}$.*

Definition 2.2.14 (Formula Denotation) *The denotation $[\varphi]^{\mathfrak{I}}_{\mathcal{V}} \in \{\textbf{\textit{false}}, \textbf{\textit{true}}\}$ of a formula $\varphi \in For_{\Sigma}$ is defined as follows:*

- *If $\varphi = p(t_1, \ldots, t_n)$ is an atomic formula with $a(p) = n$ and $t_1, \ldots, t_n \in T_{\Sigma}$ then*
$$[\varphi]^{\mathfrak{I}}_{\mathcal{V}} = p^{\mathfrak{I}}([t_1]^{\mathfrak{I}}_{\mathcal{V}}, \ldots, [t_n]^{\mathfrak{I}}_{\mathcal{V}});$$
- *If $\varphi = \neg\phi$ then* $[\varphi]^{\mathfrak{I}}_{\mathcal{V}} = \begin{cases} \textbf{\textit{true}} & \textit{iff. } [\phi]^{\mathfrak{I}}_{\mathcal{V}} = \textbf{\textit{false}}, \\ \textbf{\textit{false}} & \textit{iff. } [\phi]^{\mathfrak{I}}_{\mathcal{V}} = \textbf{\textit{true}}. \end{cases}$
- *If $\varphi = (\phi_1 \vee \phi_2)$, then*
$$[\varphi]^{\mathfrak{I}}_{\mathcal{V}} = \begin{cases} \textbf{\textit{true}} & \textit{iff. } [\phi_1]^{\mathfrak{I}}_{\mathcal{V}} = \textbf{\textit{true}}, \\ \textbf{\textit{true}} & \textit{iff. } [\phi_2]^{\mathfrak{I}}_{\mathcal{V}} = \textbf{\textit{true}}, \\ \textbf{\textit{false}} & \textit{otherwise.} \end{cases}$$
- *If $\varphi = (\forall v.\phi)$, then*
$$[\varphi]^{\mathfrak{I}}_{\mathcal{V}} = \begin{cases} \textbf{\textit{false}} & \textit{iff there is a variable assignment function } \mathcal{V}_* \\ & \textit{with respect to} D \textit{ such that } \mathcal{V}_* \textit{ may only differ from} \\ & \mathcal{V} \textit{ in the value assigned to the variable } v \textit{ and } [\phi]^{\mathfrak{I}}_{\mathcal{V}_*} = \textbf{\textit{false}}; \\ \textbf{\textit{true}} & \textit{otherwise.} \end{cases}$$

Remark 2.2.1 *The denotation of other logical connectives such as $\wedge, \rightarrow, \exists$ are given similarly, or can be reduced to the denotation of the connectives given above using the well-known tautologies. For example $\phi_1 \wedge \phi_2 \equiv \neg(\neg\phi_1 \vee \neg\phi_2)$.*

The reader may note that the value of a formula depends on the valuation function only if there are free variables in that formula. For sentences we may simply write $[\psi]_{\mathfrak{I}}$.

Definition 2.2.15 (Model) *Given an interpretation $\mathfrak{I}$, we say that $\mathfrak{I}$ satisfies a sentence φ (or that $\mathfrak{I}$ is a model), if $[\varphi]_{\mathfrak{I}} = \textbf{\textit{true}}$. If Φ is a set of sentences, we also say that $\mathfrak{I}$ satisfies Φ, if $\mathfrak{I}$ satisfies all sentences $\varphi \in \Phi$.*

Definition 2.2.16 (Entailment) *We call a sentence φ a* logical consequence *of a set of sentences Φ ($\Phi \models \varphi$) iff every model of Φ is also a model of φ. In such a case, we also say that Φ* entails *φ (or φ follows from Φ).*

Definition 2.2.17 (Satisfiability) *A set of sentences Φ is called satisfiable (consistent) if it has at least one model.*

Definition 2.2.18 (Tautology) *A sentence φ is called a tautology (or valid sentence) if it is satisfied in all interpretations, i.e. it is a logical consequence of the empty set of sentences: $\models \varphi$.*

2.2.3. Normal Forms

Certain subsets of FOL, e.g. Logic Programming or Description Logics, only allow FOL sentences that fulfill certain syntactic conditions. This subsection lists structural forms of FOL formulae that are important in our context.

Definition 2.2.19 (Substitution) *Let φ be a formula, x be a variable and t be a term. $\varphi[x/t]$ is a formula, which is produced from φ by* substituting *the term t for every free variable x in φ.*

Definition 2.2.20 (Prenex normal form (PNF)) *A formula $\varphi \in For_\Sigma$ is in prenex normal form if it is of the form*

$$\varphi \equiv Q_1 v_1 \ldots Q_n v_n . M$$

where each $Q_i \in \{\forall, \exists\}$ and M (matrix) is a quantifier free formula. $\mathbf{Q} = \{Q_1, \ldots, Q_n\}$ is usually called prefix.

Definition 2.2.21 (Conjunctive normal form (CNF)) *A formula $\varphi \in For_\Sigma$ in prenex normal form is in conjunctive normal form, if it is a conjunction consisting of one or more conjuncts, each of which is a disjunction of one or more literals $l_{i,j} \in L_\Sigma$:*

$$\varphi \equiv Q_1 v_1 \ldots Q_n v_n . (l_{1,1} \vee \ldots \vee l_{1,n_1}) \wedge \ldots \wedge (l_{k,1} \vee \ldots \vee l_{k,n_k})$$

Definition 2.2.22 (Skolem standard form (SSF)) *A formula $\varphi \in For_\Sigma$ in prenex normal form is in Skolem standard form (or universal form) if the prefix does not contain any existential quantifiers:*

$$\varphi \equiv \forall_1 v_1 \ldots \forall_n v_n . M$$

The proof of the following lemma is given in (Cremers et al., 1994)[pp. 35-36].

Lemma 2.2.1 *Every formula $\varphi \in For_\Sigma$ can be translated into an equivalent formula, which is in CNF.*

Every sentence φ in prenex normal form can be translated into a formula ψ in Skolem standard form, where φ is satisfiable iff ψ is satisfiable.

The translation of a formula φ into Skolem standard form is also called *skolemization* and can be carried out mechanically by Algorithm 2.1.

Require:
A formula φ in PNF.
while φ contains $\exists$ **do**
$\varphi = \forall y_1 \forall y_2 \ldots \forall y_n \exists z.\psi$ for a formula ψ in PNF and $n \geq 0$;
Let $f(y_1, \ldots, y_n)$ be a new n-ary function symbol, which is not in ψ;
Let $\varphi = \forall y_1 y_2 \ldots y_n.\psi[z/f(y_1, \ldots, y_n)]$;
end while
Ensure: φ

Algorithm 2.1: Skolemization algorithm

Definition 2.2.23 (Clause) *A clause is the universal closure of a disjunction of literals $L_i \in L_\Sigma$ with $i \in [1, n]$:*

$$\forall_1 v_1 \ldots \forall_n v_n.(L_1 \vee \ldots \vee L_n)$$

Definition 2.2.24 (Clause Normal Form) *A sentence $\varphi \in For_\Sigma$ is in clause normal form, if it is a conjunction of clauses.*

Lemma 2.2.2 *A sentence $\varphi \in For_\Sigma$ is satisfiable iff. its clause normal form is satisfiable.*

The lemma follows directly from lemma 2.2.1 on page 18.

Since all variables in clauses are universally qualified, we will omit the quantifiers if appropriate. We usually sort the literals $L_i \in L_\Sigma$ and write positive literals on the left hand side, and all negative literals on the right hand side of a clause:

$$L_1 \vee \ldots \vee L_k \vee \neg L_{k+1} \vee \ldots \vee \neg L_n$$

A simple transformation leads to:

$$L_1 \vee \ldots \vee L_k \leftarrow L_{k+1} \wedge \ldots \wedge L_n$$

Definition 2.2.25 (Gentzen formula) *A clause of the form*

$$P_1 \vee \ldots \vee P_k \leftarrow N_1 \wedge \ldots \wedge N_n$$

where $P_i, N_i \in L_\Sigma$ is called Gentzen formula.

Definition 2.2.26 (Horn formula) *A Gentzen formula with $k \leq 1$ is called Horn formula.*

$(P_1, N_i \in L_\Sigma)$	Formula	Name
$k = 1, n \geq 1$	$P_1 \leftarrow N_1 \wedge \ldots \wedge N_n$	definite program clause (*rule*)
	P_1 **:-** $N_1, \ldots, N_n$.	(Prolog notation)
$k = 1, n = 0$	$P_1 \leftarrow$	(positive) unit clause (*fact*)
	P_1.	(Prolog notation)
$k = 0, n \geq 1$	$\leftarrow N_1 \wedge \ldots \wedge N_n$	definite goal clause (*query*)
	:- $N_1, \ldots, N_n$	(Prolog notation)

Table 2.1.: Terminology for Different Types of Clauses

2.3. Logic Programming

Logic Programming environments typically offer several proprietary constructs with built-in semantics, which can be ignored for our purpose. This leads to the presentation offered here, which is a purely declarative one that basically understands logic programs as a finite set of horn formulae.

2.3.1. Syntax

Logic Programs consist of the three different types of Horn clauses that are presented in Table 2.1 on page 20. In the following we will use Prolog-style notation. Implications $\leftarrow$ will be written as **:-**, conjunctions $\wedge$ are written as ",“ and variables will be distinguished from constants by their capitalization. Each clause ends with a ".“.

Definition 2.3.1 (Program Clause) *A program clause wrt. to Σ is a Horn formula of the form*

$$P_1 \textbf{ :- } N_1, \ldots, N_n.$$

where the literals $P_1, N_i \in L_\Sigma$ with $i \in [0, n]$.

We distinguish three different types of clauses, which are presented in Table 2.1.

Definition 2.3.2 (Logic Program) *Given a signature Σ, a logic program $\mathcal{LP}$ is defined as a finite set of program clauses wrt. Σ.*

The literal P_1 in a definite program clause is called rule head (or left-hand side). The conjunction of literals N_i in a definite program clause is called rule body (or right-hand side).

2.3.2. Semantics

Several possibilities to define the semantics of Logic Programs exist. We will present a model theoretic approach and fixpoint semantics. The reader may note that additional possibilites exist, e.g. SLD-resolution which is also a basis for evaluating logic programs.

2.3.2.1. Herbrand Models

If we choose model theory as an utility for defining the semantics of Logic Programs, we have to choose among an infinite number of possible structures. In order to limit search for possible models to certain structures, we use the Herbrand theory which imposes a syntactic restriction on the admissible structures. We will define how the domain of Herbrand structures looks like and how terms are interpreted.

Definition 2.3.3 (Herbrand universe) *The domain Δ^H of a Herbrand structure for a signature $\Sigma = (C, F, P, V, a)$ is called Herbrand universe and is inductively defined as follows:*

- $c \in \Delta^H$ *for all* $c \in C$*;*
- $f(t_1, \ldots, t_n) \in \Delta^H$ *if* $f \in F$ *and each* $t_i \in \Delta^H$

Consequently, the Herbrand universe is just the set of all ground terms that can be built from the constant symbols and function symbols in the given signature Σ.

Definition 2.3.4 (Herbrand interpretation) *A Herbrand interpretation $\mathcal{H} = (\Delta^H, \cdot^H)$ consists of the Herbrand universe Δ^H and an interpretation function $\cdot^H : C \cup F \rightarrow \Delta^H$ assigns the constants and functions of a signature $\Sigma = (C, F, P, V, a)$, to Δ^H, such that they are assigned to "themselves", i.e. $[c]^H_{\mathcal{V}} = c$ and $[f(t_1, \ldots, t_n)]^H_{\mathcal{V}} = f([t_1]^H_{\mathcal{V}}, \ldots, [t_n]^H_{\mathcal{V}})$.*

Definition 2.3.5 (Herbrand base) *Given the Herbrand universe Δ^H for a logic program $\mathcal{LP}$, the Herbrand base B_H is the set of all ground atoms (ground literals without negation) that can be formed from predicate symbols in the logic program $\mathcal{LP}$ and terms in Δ^H.*

Definition 2.3.6 (Herbrand model) *A Herbrand interpretation H which satisfies $\mathcal{LP}$ is called Herbrand model for a logic program $\mathcal{LP}$. It can be identified with the subset of the Herbrand base B_H for which each ground atom is valuated to true, i.e. $[p(t_1, \ldots, t_n)]^H_{\mathcal{V}} =$ **true**.*

The following proposition is shown in (Schöning, 1995)[pp. 82-83]:

Proposition 2.3.1 *A sentence φ in skolem standard form is only satisfiable if it has a Herbrand model.*

Lemma 2.3.1 *A logic program has a model if and only if it has a Herbrand model.*

The lemma follows directly from proposition 2.3.1.

2.3.2.2. Fixpoint Semantics

Fixpoint semantics is an alternative way to define the semantics of logic programs. Fixpoint semantics utilizes a function $T_{\mathcal{LP}}$ which takes the set ground facts in $\mathcal{LP}$ and computes a new set of facts by applying the definite program clauses of $\mathcal{LP}$.

(Ceri et al., 1990)[p. 105] shows the following proposition:

Proposition 2.3.2 (Fixpoints are Herbrand models) *If $M \subseteq B_H$ is a fixpoint of $T_{\mathcal{LP}}$ (i.e. $T_{\mathcal{LP}}(M) = M$), then M is a model of $\mathcal{LP}$.*

We can understand a logic program $\mathcal{LP}$ as definition of an operator $T_{\mathcal{LP}} : 2^{B_H} \rightarrow 2^{B_H}$, where 2^{B_H} denotes the set of all (possible) Herbrand interpretations of $\mathcal{LP}$. Different definitions for this operator, which is also called the immediate consequence operator, exist in the literature, e.g. in (Abiteboul et al., 1995; Ceri et al., 1990); intuitively, it yields all atoms that can be derived by a single application of some rule in $\mathcal{LP}$ given the ground facts of $\mathcal{LP}$. All definitions of $T_{\mathcal{LP}}$ presented in the literature are monotone and continuous. This has interesting consequences:

The Knaster-Tarski theorem (Tarski, 1955) states that every monotone fixpoint function has a least fixpoint. Consequently, $T_{\mathcal{LP}}$ has a least fixpoint $T^{\infty}_{\mathcal{LP}}$, since $T_{\mathcal{LP}}$ is a monotone function. Moreover, since $T_{\mathcal{LP}}$ is also continuous, by Kleene's Theorem $T^{\infty}_{\mathcal{LP}}$ is the limit of the sequence $T^{0}_{\mathcal{LP}} = \emptyset, T^{i+1}_{\mathcal{LP}} = T_{\mathcal{LP}}(T^{i}_{\mathcal{LP}}), i \geq 0$.

It can also be shown that the minimal Herbrand model of a logic program $\mathcal{LP}$ is the least fixpoint of $T_{\mathcal{LP}}$. Consequently, we can compute the minimal Herbrand model by a fixpoint iteration. Moreover, the fixpoint is always reached within a finite number of iteration steps for positive logic programs without function symbols, i.e. Datalog programs.

2.3.3. Variants and Extensions

2.3.3.1. Datalog

Datalog[1] is a Logic Programming language, which two restrictions on rules and syntax. Firstly, its signature is impoverished by only permitting predicate symbols and constants. Secondly, all Datalog rules must be safe.

Definition 2.3.7 (Datalog Signature) *A Datalog signature Σ_D is a 4-tuple (C, P, V, a) and consists of*

- *a set C of constant symbols,*
- *a set P of predicate symbols,*
- *a set V of variable symbols*
- *and a function $a : P \to \mathbb{N}$.*

Definition 2.3.8 (Safe rule) *A rule is safe, if all variables in the rule head also appear in the rule body; in particular, all unit clauses are ground.*

The restrictions are motivated by the fact that all models of Datalog programs are finite if the number of constants is finite. The finiteness is due to the absence of function symbols and the safety of rules.

Definition 2.3.9 (Datalog Program) *Given a signature Σ_D, a Datalog program $\mathcal{LP}_D$ is defined as a finite set of safe program clauses wrt. Σ_D.*

Definition 2.3.10 (Intensional, extensional predicates) *The set P in the signature Σ_D is partitioned into two sets $\mathbf{P}_{edb}$ and $\mathbf{P}_{idb}$ such that $P = \mathbf{P}_{edb} \cup \mathbf{P}_{idb}$ and $\mathbf{P}_{edb} \cap \mathbf{P}_{idb} = \emptyset$.*

The elements of $\mathbf{P}_{edb}$ denote extensionally defined predicates, i.e. predicates that are effectively used to assert the facts of the logic program. The elements of $\mathbf{P}_{idb}$ denote intensionally defined predicates, i.e. predicates which are defined by appearing in the head of rules. The extension of $\mathbf{P}_{idb}$ predicates is defined implicitly and is derived from the extensional predicates by interpretation of the $\mathcal{LP}$.

[1] It is difficult to attribute Datalog to particular researchers because it is a restriction or extension of many previously proposed languages; some of the early history is discussed in (Maier & Warren, 1988).

Definition 2.3.11 (Extensional Database) *The extensional database EDB is the set of all predicates that occur only in the body of rules of $\mathcal{LP}$.*

Definition 2.3.12 (Intensional Database) *The intensional database IDB is the set of all predicates that occur in the head of rules of $\mathcal{LP}$.*

2.3.3.2. Equivalence

Many Logic Programming systems provide a built-in equivalence predicate to capture the equivalence of terms. The semantics of this predicate can be provided by an equivalence theory, that is usually built into the system. For convenience, we write the predicate infix instead of the usual prefix notation for predicates.

Definition 2.3.13 (Equivalence theory) *Let $\Sigma = (C, F, P, V, a)$ be a signature and $\mathcal{LP}$ be a logic program. An equivalence theory $=_\Sigma$ consists of a dedicated set of formulas that capture the five axioms of equivalence.*

- *Reflexivity:* $X = X$:-.
- *Symmetry:* $X = Y$:- $Y = X$.
- *Transitivity:* $X = Z$:- $X = Y, Y = Z$.

Substitutivity ensures the correct semantics of equivalence with respect to function and predicate symbols:

- *Functions: For all $f \in F$:*

$$f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \text{ :- } x_1 = y_1, \ldots, x_n = y_n.$$

- *Predicates: For all $p \in P$:*

$$p(x_1, \ldots, x_n) \text{ :- } p(y_1, \ldots, y_n), x_1 = y_1, \ldots, x_n = y_n.$$

(Hoelldobler, 1987) shows that such an axiomatization renders the algebraic semantics of an equivalence relation. The reader may note that the axioms of substitutivity have to be instantiated for all predicates p and functions f used in the logic program. Other extensions of Logic Programming, which provide built-in predicates for inequality or arithmetic predicates are defined similarly.

2.3.3.3. Negation

The idea of adding negation to Logic Programming led to a flurry of research in the mid-eighties and mid-nineties. This is due to the fact, that adding negation to Logic Programs is not straightforward, i.e. it leads to non-monotonic reasoning. (Apt & Bol, 1994) present a detailed survey and comparison of the different proposals presented in the literature.

We will only consider the standard approach to extend Logic Programs with negation. We add a new operator $\neg^*$ whose semantics differs from the semantics for first-order negation. The assigned semantics is based on the principle of negation as failure. This principle states that $\neg\phi$ is a consequence of $\mathcal{LP}$ if ϕ cannot be proven from $\mathcal{LP}$ by any derivation. Thereby we implicitly apply the closed world assumption (CWA) which states that for any ground atom ϕ, $\neg\phi$ is a consequence of $\mathcal{LP}$ if ϕ cannot be proved from $\mathcal{LP}$.

Definition 2.3.14 (Normal program clause) *A normal program clause with respect to a signature Σ is a formula of the form H :- $B_1 \wedge \ldots \wedge B_n$, where $H \in L_\Sigma$ and B_i are either positive literals ($B_i = L_j$) or negated literals ($B_i = \neg^* L_j$) and all $L_j \in L_\Sigma$.*

Definition 2.3.15 (Normal Logic Program) *Given a signature Σ, a normal program $\mathcal{LP}^*$ is defined as a finite set of normal program clauses with respect to Σ.*

The terminology for normal logic programs, e.g. fact, rule, etc., is defined analogously to (standard) logic programs.

The semantics of normal logic programs departs from the semantic of positive logic programs due to the introduction of $\neg^*$. For example, it is no longer provided that any (single) minimal Herbrand model exists for normal logic programs.

Example 2.3.1 *Consider the following programs:*

- $\mathcal{LP}_1 = \{p$:- $\neg^* p.\}$ *has no fixpoint.*
- $\mathcal{LP}_2 = \left\{ \begin{array}{l} p \text{ :- } \neg^* q. \\ q \text{ :- } \neg^* p. \end{array} \right\}$ *has two minimal fixpoints: $\{p\}$ and $\{q\}$.*

However, there are classes of logic programs for which single minimal Herbrand models exist. These classes of logic programs have a syntactic characterization based on predicate dependencies.

2.3.3.4. Stratifiable Programs

We start our presentation with so-called semi-positive programs.

Definition 2.3.16 (Semipositive Programs) *A normal program $\mathcal{LP}^*$ is semipositive if, whenever a negative literal $\neg^* r(x_1, \ldots, x_n)$ occurs in the body of a rule in $\mathcal{LP}^*$, $r \in \mathbf{P}_{edb}$.*

As the name suggests semipositive programs are almost positive. One could eliminate negation from semipositive programs by adding a new predicate $\overline{r}$ to each $r \in \mathbf{P}_{edb}$ holding the complement of r and replacing each $\neg^* r$ by $\overline{r}$. Thus it is not surprising that semipositive programs have a unique minimal Herbrand model.

Stratifiable programs are a natural extension to semipositive programs. Here, the use of negation is no longer restricted to EDB predicates, but limited to some defined predicates (views) whose definition is independent from rules using the predicate. Consequently such views can be treated as if they were EDB predicates. Unfortunately stratification does not provide semantics for all normal logic programs, e.g. those presented in Example 2.3.1 on page 25, but many normal logic programs can be transformed into an equivalent stratified logic program by a process called stratification.

Definition 2.3.17 (Stratification) *A stratification of a normal program $\mathcal{LP}^*$ over a signature Σ is a sequence of normal logic programs $\mathcal{LP}^*_1, \ldots, \mathcal{LP}^*_n$ such that for some mapping ρ from $\mathbf{P}_{idb}$ to $[1 \ldots n]$:*

1. *$\{\mathcal{LP}^*_1, \ldots, \mathcal{LP}^*_n\}$ is a partition of $\mathcal{LP}^*$;*
2. *For each predicate $P \in \mathbf{P}_{idb}$, all rules in $\mathcal{LP}^*$ whose head is P are in $\mathcal{LP}^*_{\rho(P)}$, i.e. in the same program of the partition;*
3. *If $P(u)$ **:-** $\ldots P'(v) \ldots$. is a rule in $\mathcal{LP}^*$ and $P' \in \mathbf{P}_{idb}$, then $\rho(P') \leq \rho(P)$;*
4. *If $P(u)$ **:-** $\ldots \neg^* P'(v) \ldots$. is a rule in $\mathcal{LP}^*$ and $P' \in \mathbf{P}_{idb}$, then $\rho(P') < \rho(P)$.*

Each $\mathcal{LP}^*_i$ is called a stratum of the stratification, and ρ is called the stratification mapping.

There is a simple test for checking if a program is stratifiable, which involves testing for an acyclicity condition in the definitions of predicates using negation. This is achieved via a rule precedence graph.

Definition 2.3.18 (Rule Precedence Graph) *The rule precedence graph $G_{\mathcal{LP}} = (V, E, \{+, -\})$ of a logic program $\mathcal{LP}$ over a signature Σ is the labelled graph where $V = \mathbf{P}_{idb}$ and*

- *if* $P(u)$ *:-* $\ldots Q(v) \ldots$ *then* $(P, Q) \in E$ *and labelled with* $+$ *(positive edge),*
- *if* $P(u)$ *:-* $\ldots \neg^* Q(v) \ldots$ *then* $(P, Q) \in E$ *and labelled with* $-$ *(negative edge).*

(Abiteboul et al., 1995)[p. 380] show the following proposition:

Proposition 2.3.3 (Stratifiability of logic programs) *A logic program $\mathcal{LP}$ is stratifiable iff its precedence graph $G_{\mathcal{LP}}$ has no cycle containing a negative edge.*

The semantics of stratified programs is obtained by applying, in order, the semantics of programs of $\mathcal{LP}_i^*$. This provides the semantics of a logic program under stratification ρ.

Several other semantics have been proposed for normal logic programs, but are not used in this dissertation.

2.3.4. Reasoning Problems

Logic Programming supports one basic type of reasoning with respect to a logic program $\mathcal{LP}$, namely answering *Atom Queries*. Atom queries come in two variants. Firstly, *Open Atom Queries* determine all substitutions (variable bindings) for all variables in a goal clause (query) A. Secondly, *Ground Atom Queries* [2] determine whether a ground atomic formula A is entailed.

2.3.5. Complexity

We briefly discuss the different complexity results known for Logic Programming. Our discussion is based on the survey of (Dantsin et al., 2001). Naturally, different complexity results can be obtained for the variants and extensions of Logic Programming presented in Section 2.3.3 on page 23. Three main kinds of complexity connected to Logic Programming can be distinguished (Vardi, 1982).

Data complexity is the complexity of checking whether $EDB \cup \mathcal{LP} \models A$ when logic programs $\mathcal{LP}$ are fixed, whereas input databases EDB and ground atoms A are an input.

Program complexity is the complexity of checking whether $EDB \cup \mathcal{LP} \models A$ when input databases EDB are fixed whereas Logic Programs $\mathcal{LP}$ and ground atoms A are an input.

[2] Ground atom queries are actually a special case of open atom queries.

Combined complexity is the complexity of checking whether $EDB \cup \mathcal{LP} \models A$ where input databases EDB, Logic programs $\mathcal{LP}$ and ground atoms A are an input.

2.3.5.1. Datalog

For plain Datalog, as well as for all other versions of Datalog considered in the thesis, the combined complexity is equivalent to the program complexity with respect to polynomial-time reductions. This is due to the fact that each $\mathcal{LP}$ can be easily rewritten into a program where all facts are encoded into rules (Dantsin et al., 2001).

Datalog is data complete for the complexity class P and program complete for EXPTIME (Vardi, 1982; Immerman, 1986). Moreover, Datalog precisely expresses the complexity class P on *ordered databases* [3]. If databases are not ordered, this result does not hold. For example, one can write no Datalog program that determines whether an even number of facts exists for a predicate p, which can easily be determined by a polynomial algorithm.

2.3.5.2. Datalog with negation

Since the stratification algorithm is a polynomial algorithm, stratified Datalog with negation has the same complexity as plain Datalog, i.e. it is data complete for P and program complete for EXPTIME (Implicit in (van Gelder, 1989; van Gelder et al., 1991)).

2.3.5.3. Logic Programming

A classical result that can be taken from fixpoint theory is that Logic Programming can express all recursively enumerable predicates, i.e. it is undecidable. Similarly a reduction of Logic Programming to deterministic Turing machines is possible and can be used to show that Logic Programming is computationally complete (Andeka & Nemeti, 1978).

2.3.6. Evaluation Strategies

Unfortunately the model theoretic semantics gives us no hint on how to compute answers to queries or how to entail implicit information. Moreover, the above definitions do not even provide a reasonable method for testing whether a given

[3] An ordered database contains a special binary relation *succ* that provides a successor relation on all constants in Δ^H.

ground fact ϕ is entailed by a logic program $\mathcal{LP}$. This is due to three types of infinity:

1. We have to check a infinite number of possible Herbrand models,
2. Many of the Herbrand models are infinite,
3. If a rule contains variables we must consider an infinite number of possible variable substitutions.

Therefore evaluation algorithms typically compute the minimal Herbrand model. Approaches to evaluation of logic programs fall into the two main categories of Bottom-Up and Top-Down strategies.

2.3.6.1. Bottom-Up Strategies

Bottom-Up strategies typically compute the minimal Herbrand model starting with ground facts and work their way upwards through a proof tree deriving new facts until no new facts can be derived. Thereby, rules are seen as productions and are processed forward in direction of implications. Therefore, these strategies are also called *Forward-Chaining* strategies.

In this sense, we can consider fixpoint semantics as an prototypical bottom-up approach in the sense that a (naïve) forward-chaining implementation can directly be derived from the definition $T_{\mathcal{LP}}$. Several other bottom-up strategies are well known, e.g. naive, semi-naive, and Henschen-Naqvi methods. Detailed accounts on these methods are given in (Ceri et al., 1990)[Chapter 9].

In practice, generating the minimal Herbrand model is often cumbersome, since, even in the case of Datalog it is in general exponential in the size of the $\mathcal{LP}$. Moreover, it is not always necessary to compute the whole model in order to determine whether $\mathcal{LP} \models A$ for some particular fact A. Therefore a series of optimizations for bottom-up strategies were proposed. In our context only one is relevant, the so-called Magic Sets method.

The purpose of the *Magic Sets* (Ramakrishnan, 1991) method is the optimization of Datalog programs with particular adornments of the goal predicates. It is a logical rewriting method, viz. it transforms a given program into another equivalent program. The transformation is based on the idea of *Sideways Information Passing* (*SIP*). Intuitively, given a certain rule and a subgoal in the rule body with some bound arguments, one can solve the subgoal and obtain bindings for uninstantiated variables in other argument positions. These bindings can then be transferred to other

subgoals in the rule body and they, in their turn, transmit bindings to other variables. This is the normal behavior of top-down evaluation strategies, which are presented next. Therefore, the Magic Sets method can also be described as an attempt to incorporate the binding passing strategy of top-down methods into bottom-up methods.

2.3.6.2. Top-Down Strategies

Top-Down Strategies are completely different strategies of deriving new facts from a logic program. They start from the query (goal clause) and work their way down to facts in a top-down fashion. In a sense, the rules of the logic program are seen as problem generators and each goal (query) is seen as a problem that must be solved. These strategies, also known as *Backward-Chaining* strategies, are based on variants of the famous *Resolution Principle* of Robinson (Robinson, 1965) and build linear refutation trees. The major variant is *SLD-Resolution* [4] (Kowalski & Kuehner, 1971). It is important to note that SLD-resolution does not stand for a particular algorithm, but for an entire class of algorithms, which particularly differ in their proof tree traversal methods, e.g. breath-first or depth-first, and halting conditions.

Roughly, SLD-resolution can be described as follows. A goal is a conjunction of atoms, and a substitution is a function σ that that maps variables $x_1, \ldots, x_n$ to terms $t_1, \ldots, t_n$. The result of simultaneous replacement of variables x_i by terms t_i in an expression E is denoted by $E\sigma$. For a given goal G and a program $\mathcal{LP}$, SLD-resolution tries to find a substitution σ such that $G\sigma$ logically follows from $\mathcal{LP}$. The initial goal is repeatedly transformed until the empty goal is obtained. Thus, this transformation performs a relevant optimization, because the computation automatically disregards many of the facts which are not useful for producing the result. Each transformation step is based on the application of the resolution rule to a selected atom B_i from the goal $B_1, \ldots, B_m$ and a clause $A_0 \texttt{:-} A_1, \ldots, A_n$ from $\mathcal{LP}$. SLD-resolution tries to unify B_i with the head A_0, that is, to find a substitution σ such that $A_0\sigma = B_i\sigma$. Such a substitution σ is called a unifier of A_0 and B_i . If a unifier σ exists, a most general such σ (which is essentially unique) is chosen and the goal is transformed into

$$(B_1, \ldots, B_{i-1}, A_1, \ldots, A_n, B_{i+1}, \ldots, B_m)\sigma.$$

A more detailed account of SLD-resolution and its extension for dealing with negation as failure (SLDNF) is given in (Lloyd, 1987).

[4] SLD is a shorthand for Linear Resolution with Selection Function for Definite Clauses.

2.4. Description Logics

Description Logics have initially been designed to fit object-centric knowledge representation formalisms like semantic networks and frame systems with a formal and declarative semantics. During the 20 years of research in this field of knowledge representation a family of logics has evolved, which can be distinguished from each other by the constructors and axioms available in each language. Generally, the particular selection of constructors and axioms is made such that inferencing with the logic is decidable.

Consequently, most description logics turnout to be subsets or variants of $\mathcal{C}^2$, the subset of first-order logic extended with counting quantifiers, where formulae with no function symbols and maximum two variables, which is known to be decidable (Graedel et al., 1997). Description Logics can therefore also be understood as an attempt to address the major drawbacks of using first-order logic for knowledge representation, since all relevant inference problems in FOL are undecidable and the epistemic neutrality of FOL leads to a rather cumbersome syntax, which is addressed by DL by a variable-free notation.

2.4.1. Syntax

Description Logics are formalisms that support the logical description of classes and properties. Consequently, the basic elements of description logics are so-called classes,[5] which group objects into categories and properties,[6] which relate pairs of objects with each other. An arbitrary class and property *Description* can be constructed from two disjoint sets of symbols, *Class Names* and *Property Names* (also called *Atomic Classes* and *Atomic Properties*) using a variety of class- and property-forming constructors, the range of which is specific for the particular description logic (cf. Table 2.2 on page 32). $\mathcal{AL}$ (Schmidt-Schauß & Smolka, 1991) is usually considered to be the basic description logic.

Example 2.4.1 (Childless Man) *Given the atomic classes* PERSON *and* FEMALE *and the atomic property* HASCHILD *we can describe a childless man in* $\mathcal{AL}$ *as*

$$\text{PERSON} \sqcap \neg\text{FEMALE} \sqcap \forall\text{HASCHILD}.\bot.$$

[5] In the DL literature, the term *Concept* is used more often. We use the term *Class* as it is commonly used in the Web ontology context.

[6] Again, the DL literature would use the term *Role* instead of the term *Property* as it is done in the Web context.

[7] In some DLs Qualified number restrictions on properties are available, e.g. in DAML+OIL, hence the number restriction is augmented with a universal quantification on the range of the property.

Name	Syntax	Semantics	Symbol
Atomic named class	A	$A^{\mathfrak{I}}$	$\mathcal{AL}$ $(\mathcal{S})$
Universal class (Top)	$\top$	$\Delta^{\mathfrak{I}}$	$\mathcal{AL}$ $(\mathcal{S})$
Empty class (Bottom)	$\bot$	$\emptyset$	$\mathcal{AL}$ $(\mathcal{S})$
Atomic complement	$\neg A$	$\Delta^{\mathfrak{I}} \backslash A^{\mathfrak{I}}$	$\mathcal{AL}$ $(\mathcal{S})$
Conjunction (And)	$C \sqcap D$	$C^{\mathfrak{I}} \cap D^{\mathfrak{I}}$	$\mathcal{AL}$ $(\mathcal{S})$
Value restriction	$\forall R.C$	$\{a \in \Delta^{\mathfrak{I}} \mid \forall b((a,b) \in R^{\mathfrak{I}} \rightarrow b \in C^{\mathfrak{I}})\}$	$\mathcal{AL}$ $(\mathcal{S})$
Limited existential restriction	$\exists R.\top$	$\{a \in \Delta^{\mathfrak{I}} \mid \exists b((a,b) \in R^{\mathfrak{I}})\}$	$\mathcal{AL}$ $(\mathcal{S})$
Disjunction	$C \sqcup D$	$C^{\mathfrak{I}} \cup D^{\mathfrak{I}}$	$\mathcal{U}(\mathcal{S})$
Full complement	$\neg C$	$\Delta^{\mathfrak{I}} \backslash C^{\mathfrak{I}}$	$\mathcal{C}(\mathcal{S})$
Full existential restr.	$\exists R.C$	$\{a \in \Delta^{\mathfrak{I}} \mid \exists b((a,b) \in R^{\mathfrak{I}} \wedge b \in C^{\mathfrak{I}})\}$	$\mathcal{E}(\mathcal{S})$
Unqualified[7] number restr.	$\geq n\, R$	$\{a \in \Delta^{\mathfrak{I}} \mid \mid \{b \in \Delta^{\mathfrak{I}} \mid (a,b) \in R^{\mathfrak{I}}\}\mid \geq n\}$	$\mathcal{N}$
	$\leq n\, R$	$\{a \in \Delta^{\mathfrak{I}} \mid \mid \{b \in \Delta^{\mathfrak{I}} \mid (a,b) \in R^{\mathfrak{I}}\}\mid \leq n\}$	
Enumeration	$\{i_1, \ldots, i_n\}$	$\{i_1^{\mathfrak{I}}, \ldots i_n^{\mathfrak{I}}\}$	$\mathcal{O}$
Inverse property	R^-	$\{(b,a) \in \Delta^{\mathfrak{I}} \times \Delta^{\mathfrak{I}} \mid (a,b) \in R^{\mathfrak{I}}\}$	$\mathcal{I}$
Transitive property	R^+	$\bigcup_{n \geq 1} (R^{\mathfrak{I}})^n$	$\mathcal{R}^+(\mathcal{S})$

Table 2.2.: DL Class and Property Constructors

Many applications, e.g. reasoning about database schemas (Calvanese et al., 1998) or ontological engineering (Doyle & Patil, 1991) are supposed to require more expressive logics than $\mathcal{AL}$. Hence, several approaches to extending $\mathcal{AL}$ have been taken in the literature. The two most prominent possibilities are providing additional constructors for forming classes and properties, prominent examples for such additions are listed in Table 2.2,[8] and formulating restrictions on the interpretation of properties. (Schmidt-Schauß & Smolka, 1991) suggest a naming scheme in which a letter or symbol is assigned to each extension of $\mathcal{AL}$ and written after the starting $\mathcal{AL}$. For example, the extension of $\mathcal{AL}$ with transitive roles (R^+) and full class complement ($\mathcal{C}$) is called $\mathcal{ALC}_{R^+}$. The $\mathcal{ALC}_{R^+}$ logic is often abbreviated as $\mathcal{S}$.[9] Some of the logics are equivalent to each other, for example $\mathcal{ALC}$ and $\mathcal{ALUE}$, since both disjunction and full existential quantification can be expressed with the full complement in combination with other constructors available in $\mathcal{AL}$ and vice versa.

[8] This table is by no means exhaustive and captures only the proposed extensions that are relevant for this dissertation.

[9] This is due to its vague correspondence with the propositional (multi) modal logic $S4_{(m)}$ (Schild, 1991).

2.4.1.1. Axioms

A description logic *knowledge base* usually consists of a set of axioms (cf. Table2.3), which can be distinguished into terminological axioms (building the so-called *TBox* $\mathcal{T}$) and assertional axioms or assertions (constituting the *ABox* $\mathcal{A}$).

Name	Syntax	Semantics	Symbol
TBox:			
Class Equivalence	$C \equiv D$	$C^{\mathfrak{I}} = D^{\mathfrak{I}}$	
Class Subsumption	$C \sqsubseteq D$	$C^{\mathfrak{I}} \subseteq D^{\mathfrak{I}}$	
Property Equivalence	$P \equiv R$	$P^{\mathfrak{I}} = R^{\mathfrak{I}}$	
Property Subsumption	$P \sqsubseteq R$	$P^{\mathfrak{I}} \subseteq R^{\mathfrak{I}}$	$\mathcal{H}$
ABox:			
Individual assertion	$C(i)$	$i^{\mathfrak{I}} \in C^{\mathfrak{I}}$	
Property filler	$R(a, b)$	$(a^{\mathfrak{I}}, b^{\mathfrak{I}}) \in R^{\mathfrak{I}}$	
Individual equivalence	$i = j$	$i^{\mathfrak{I}} = j^{\mathfrak{I}}$	
Individual inequivalence	$i \neq j$	$i^{\mathfrak{I}} \neq j^{\mathfrak{I}}$	

Table 2.3.: DL Axioms

TBox A TBox is constituted by a finite set of terminological axioms which define subsumption and equivalence relations on classes and properties (cf. Table 2.3). An equivalence axiom whose left-hand side is an atomic class (property) is called a *class (property) definition*. The respective class on the left-hand side of the equivalence axiom is called *defined* Class. Axioms of the form $C \sqsubseteq D$ for complex class descriptions C, D are called *(general) inclusion axioms*. C is called a *primitive* class, if it is atomic and occurs on the left-hand side of an inclusion axiom. Moreover, the set of axioms of the form $R \subseteq S$ where both R and S are atomic classes (properties) is called a *class (property) hierarchy*. We say that a class A *directly uses* a class B in $\mathcal{T}$ if B appears on the right-hand side of the definition of A. *Uses* is the transitive closure of the relation directly uses. We say, that a TBox $\mathcal{T}$ is *cyclic*, if any atomic class A uses itself. An example TBox is presented in Table 2.5 on page 35.

ABox Assertional axioms or *Assertions* introduce *Individuals*, i.e. instances of a class, into the knowledge base and relate individuals with each other and the introduced terminology. We can distinguish four kinds of assertions (cf. Table 2.3): *Class Assertions* express that an individual is member of a class. *Property fillers* express, that two individuals are related with each other via a given property. *Individual equivalence* assertions allows to express that two individuals are actually equivalent. Similarly, *Individual inequivalence* axioms allow to express that two individuals

are not the same. A finite set of such assertions is called *ABox* . An example ABox is presented in Table 2.5 on page 35.

Name	Syntax	Semantics
Datatype Property	T	$T^{\mathfrak{I}} \subseteq \Delta^{\mathfrak{I}} \times \Delta_D^{\mathfrak{I}}$
Datatype	d	$d^D \subseteq \Delta_D$
Datatype Negation	$\neg d$	$\Delta_D \setminus d^D$
Datatype Exists	$\exists T d$	$\{x \mid \exists y.(x,y) \in T^{\mathfrak{I}} \wedge y \in d^D\}$
Datatype Value	$\forall T.d$	$\{x \mid \forall y.(x,y) \in T^{\mathfrak{I}} \rightarrow y \in d^D\}$

Table 2.4.: Extension of a DL with a Concrete Domain

2.4.1.2. Concrete Domains

Since knowledge in Description Logics has to be represented on an abstract logical level, the possibility of reasoning on specific *Concrete Domains* such as numbers and strings is missing for many applications. In order to fill this gap (Baader & Hanschke, 1991; Pan & Horrocks, 2002) parameterize the logic with a given concrete domain and define new class constructors that allow to refer to predicates defined for the domain. More precisely, a new syntactic type called datatype properties[10] allows to attach elements of the concrete domain, e.g. natural numbers, to elements of the logical domain. Further class constructors (cf. Table 2.4) can then be used to describe constraints on the concrete data.

Example 2.4.2 (Adult) *For example, if we take a concrete domain Δ_D that provides natural numbers and an unary predicate $\geq_{18}$, we can express the condition that adults are persons of age by usage of the class* PERSON *and the datatype property* HASAGE

$$\text{ADULT} \equiv \text{PERSON} \sqcap \exists \text{HASAGE}.\ \geq_{18},$$

2.4.2. Semantics

The semantics of Description Logics is usually given in a Tarski-style model-theoretic manner. Hence, we consider interpretations $\mathfrak{I} = (\Delta^{\mathfrak{I}}, \cdot^{\mathfrak{I}})$. The non-empty set $\Delta^{\mathfrak{I}}$ serves as the domain of the interpretation, whereas the interpretation function $\cdot^{\mathfrak{I}}$ assigns a set $A^{\mathfrak{I}} \subseteq \Delta^{\mathfrak{I}}$ to every atomic class A and a binary relation $R^{\mathfrak{I}} \subseteq \Delta^{\mathfrak{I}} \times \Delta^{\mathfrak{I}}$ to every atomic property R. The interpretation function is extended

[10] (Areces & Lutz, 2002) speak of concrete features, (Pan & Horrocks, 2002) speak of concrete roles.

TBox			
$(T1)$	WOMAN	$\sqsubseteq$	PERSON
$(T2)$	MAN	$\sqsubseteq$	PERSON $\sqcap \neg$WOMAN
$(T3)$	WIFE	$\sqsubseteq$	WOMAN $\sqcap \exists$MARRIEDTO.HUSBAND
$(T4)$	HUSBAND	$\sqsubseteq$	MAN $\sqcap \exists$MARRIEDTO.WIFE
$(T5)$	FATHER	$\equiv$	MAN $\sqcap \exists$HASCHILD.PERSON
$(T6)$	MOTHER	$\equiv$	WOMAN $\sqcap \exists$HASCHILD.PERSON
$(T7)$	HASCHILD	$\sqsubseteq$	ANCESTOROF
$(T8)$	ANCESTOROF$^+$	$\sqsubseteq$	ANCESTOROF
$(T9)$	MARRIEDTO$^-$	$\sqsubseteq$	MARRIEDTO
$(T10)$	$\exists$LIVESIN.{LEIPZIG}	$\sqsubseteq$	LEIPZIGINHABITANT
$(T11)$	GENIUS $\sqcap$ COMPOSER	$\sqsubseteq$	$\forall$HASCOMPOSED.MASTERPIECE
$(T12)$	BIRTHDAYCANTATA $\sqsubseteq \exists$FOREVENT.BIRTHDAY $\sqcap$ CANTATA $\sqcap$ HOMAGE		
$(T13)$	CANTATA $\sqsubseteq$ VOCALCOMPOSITION $\sqcap$ INSTRUMENTALCOMPOSITION		
$(T14)$	VOCALCOMPOSITION $\sqsubseteq$ COMPOSITION $\sqcap \exists$FORINSTRUMENT.VOICE		
$(T15)$	INSTRUMENTALCOMPOSITION	$\sqsubseteq$	COMPOSITION $\sqcap \exists$FORINSTRUMENT.$\neg$VOICE
$(T16)$	INDYNASTY$^-$	$\sqsubseteq$	INDYNASTY
$(T17)$	ANCESTOROF	$\sqsubseteq$	INDYNASTY
ABox			
$(A1)$	$\exists$HASCHILD.MAN(JOHANN-AMBROSIUS)		
$(A2)$	COMPOSER $\sqcap$ MAN(JOHANN-SEBASTIAN)		
$(A3)$	PERSON(WILHELM-FRIEDEMANN)		
$(A4)$	HASCHILD(JOHANN-SEBASTIAN, WILHELM-FRIEDEMANN)		
$(A5)$	WOMAN(ANNA-MAGDALENA)		
$(A6)$	MARRIEDTO(JOHANN-SEBASTIAN, ANNA-MAGDALENA)		
$(A7)$	MAN(JOHANN-AMBROSIUS)		
$(A8)$	LIVESIN(JOHANN-SEBASTIAN, LEIPZIG)		
$(A9)$	GENIUS(JOHANN-SEBASTIAN)		
$(A10)$	HASCOMPOSED(JOHANN-SEBASTIAN, BWV248)		
$(A11)$	WOMAN(MARIA-BARBARA)		
$(A12)$	MARRIEDTO(JOHANN-SEBASTIAN, MARIA-BARBARA)		
$(A13)$	CANTATA(BWV248)		
$(A14)$	BIRTHDAYCANTATA(BWV213)		
$(A15)$	BIRTHDAYCANTATA(BWV214)		
$(A16)$	...		

Table 2.5.: The Bach Family Knowledge Base

to class and property constructors by inductive definition (cf. the column Semantics in Table 2.2 on page 32). We give semantics to assertions by extending the interpretation function to individual names mapping each individual a to an element $a^{\mathfrak{I}} \in \Delta^{\mathfrak{I}}$.

2.4.2.1. Terminological axioms

The semantics of axioms is defined via set relationships (cf. the column Semantics in Table 2.3 on page 33). For example, an interpretation $\mathfrak{I}$ satisfies an inclusion $C \sqsubseteq D$ if $C^{\mathfrak{I}} \subseteq D^{\mathfrak{I}}$. An interpretation satisfies (or is a model of) a TBox $\mathcal{T}$ if it satisfies each element of $\mathcal{T}$ (analogously for ABoxes $\mathcal{A}$). For this interpretation of a TBox $\mathcal{T}$, the atomic classes in $\mathcal{T}$ are divided into two sets, *Name Symbols* $\mathcal{N}_{\mathcal{T}}$, which occur on the left-hand side of some axiom and *Base Symbols* $\mathcal{B}_{\mathcal{T}}$, that occur only on the right-hand side of axioms. A *Base Interpretation* $\mathcal{J}$ interprets only base symbols. Any interpretation $\mathfrak{I}$ that interprets also name symbols is an *extension* of $\mathcal{J}$ if it has the same domain, viz $\Delta^{\mathfrak{I}} = \Delta^{\mathcal{J}}$ and if it agrees with $\mathcal{J}$ on the base symbols. The TBox $\mathcal{T}$ is *definitorial*, i.e. has unequivocal meaning, if every base interpretation $\mathcal{J}$ has only one extension. In other words, the meaning of the name symbols is completely determined.

Acyclic Tboxes Acyclic TBoxes have the nice property of being definitorial, since the definitions of name symbols $\mathcal{N}_{\mathcal{T}}$ can be understood as macros, which can be expanded out recursively such that the right-hand side of axioms in $\mathcal{T}$ only contains base symbols.

Cyclic TBoxes Generally, cyclic Tboxes $\mathcal{T}$ are not definitorial, i.e. no unique extension of the base interpretation exists. In this case, one can either assign *descriptive semantics* (cf. (Nebel, 1991)) and accept the set of possible extensions of the base interpretations as interpretations, or sort out a particular interpretation among the possible extensions. This is done in least and greatest fixpoint semantics, which view $\mathcal{T}$ as a function that associates to a name symbol A the class description C, viz. $\mathcal{T}(A) = C$. The interpretation of A is then considered as a fixpoint equation, $A^{\mathfrak{I}} = \mathcal{T}(A)^{\mathfrak{I}}$, and $\mathcal{T}$ is definitorial iff every base interpretation $\mathcal{J}$ has a unique extension that is a fixpoint of $\mathcal{T}$. *Least fixpoint semantics* would then only consider least fixpoints as admissible interpretations of $\mathcal{T}$, where $\mathfrak{I}$ of $\mathcal{T}$ is the least fixpoint (lfp) if $\mathfrak{I} \leq \mathfrak{I}'$ for every other fixpoint $\mathfrak{I}'$. Analogously *greatest fixpoint semantics* would consider only the greatest fixpoint as admissible interpretation of $\mathfrak{I}$. Unfortunately, least and greatest fixpoints must not exist for every terminology $\mathcal{T}$, therefore modern description logic systems usually assign descriptive semantics to cyclic Tboxes.

General inclusion axioms If a TBox contains general inclusion axioms, it is not definitorial, but can be made so by turning inclusion axioms into class equivalence axioms using the following rewriting:

$$C \sqsubseteq D \Leftrightarrow C \equiv \overline{C} \sqcap D$$

where $\overline{C}$ stands for the qualities of C that distinguish C from D. This rewriting is called *normalization* and it can be shown that the rewriting preserves the semantics of the TBox.

Property Hierarchy The property hierarchy obviously imposes restrictions on the interpretation of properties similar to inclusion axioms. The fact that the knowledge base may contain a property hierarchy is sometimes indicated by appending $\mathcal{H}$ to the name of the Description Logic.

2.4.2.2. Assertional axioms

$\mathfrak{I}$ maps each individual name a to an element $a^{\mathfrak{I}} \in \Delta^{\mathfrak{I}}$ in the domain of discourse. The semantics of the different ABox axioms are stated in the column semantics in Table 2.3 on page 33. For example, $\mathfrak{I}$ satisfies a class assertion $C(a)$ if $a^{\mathfrak{I}} \in C^{\mathfrak{I}}$. Similarly, when related to a TBox $\mathcal{T}$, the interpretation $\mathfrak{I}$ satisfies an assertion α (or a whole ABox $\mathcal{A}$) if it is both a model of α (or $\mathcal{A}$) and a model of $\mathcal{T}$.

2.4.2.3. Concrete domains

The semantics we assign to datatypes follows the approach of (Horrocks & Sattler, 2001):[11] To present concrete domains a Description Logic is extended with a set D of concrete datatypes, and with each $d \in D$, a set $d^D \subseteq \Delta_D$ is associated, where Δ_D is the domain of all datatypes. We will assume that:

1. the concrete domain Δ_D and the domain of interpretation $\Delta^{\mathfrak{I}}$ are disjoint,
2. a sound and complete decision procedure[12] for the emptiness of expressions of the form $d_1^D \cap \ldots \cap d_n^D$, where d_i is a (possibly negated) concrete datatype from D.

[11] This approach is simpler than the approach of (Baader & Hanschke, 1991), which allows the DL to form new datatypes. For the setting of Semantic Web applications we expect that the type system is already sufficiently structured and may include a derivation mechanism and built-in ordering relations, i.e. such as provided by the XML Schema type system.

[12] which must be provided by a separate inference component that can deal with a given concrete domain, e.g. with the arithmetics of natural numbers.

The Description Logic is additionally fitted with a new type of properties, so-called datatype properties T, and two new class constructors, datatype existential and datatype universals. The required extensions to our interpretation function are stated in Table 2.4 on page 34, for example the interpretation function $\cdot^{\mathfrak{I}}$ assigns a binary relation $T^{\mathfrak{I}} \subseteq \Delta^{\mathfrak{I}} \times \Delta_D$ to every datatype property T. Typically, property inclusion axioms of the form $T \sqsubseteq R$ (or $R \sqsubseteq T$) for a property R and a datatype property T are disallowed, since the disjointness of Δ_D and $\Delta^{\mathfrak{I}}$ leads that each model of such axioms necessarily interprets T (or R) as the empty relation.

2.4.3. Reasoning Problems

For Description Logics various reasoning tasks are usually considered. These tasks allow to draw new conclusions about the knowledge base or check its consistency. As we will see below checking the consistency is the primary reasoning problem. We will call $\mathcal{QDL}$ the language defined by the kinds of DL reasoning problems stated below.

2.4.3.1. TBox

Some reasoning tasks only consider a TBox $\mathcal{T}$:

- *Satisfiability* A class C is satisfiable with respect to $\mathcal{T}$, if a model $\mathfrak{I}$ of $\mathcal{T}$ exists, where $C^{\mathfrak{I}} \neq \emptyset$. $\mathfrak{I}$ is then also a model of C. We can check whether the whole TBox is satisfiable by asking whether $\top$ is satisfiable.
- *Subsumption* A class D subsumes a class C with respect to $\mathcal{T}$, if $C^{\mathfrak{I}} \subseteq D^{\mathfrak{I}}$ for all models $\mathfrak{I}$ of $\mathcal{T}$. We write $\mathcal{T} \models C \sqsubseteq D$. Property subsumption is defined analogously. We can distinguish two derived kinds of subsumption queries. Firstly, we can ask for the embedding of a given class C in the class hierarchy induced by $\mathcal{T}$. This problem can come in several variants. We can query for all or only the most-specific (named) superclasses of C and for all or only the most-general (named) subclasses of C in $\mathcal{T}$. Moreover, we can classify the complete TBox, i.e. compute subsumption for all pairs of classes. Property subsumption is defined analogously.
- *Equivalence* Two classes C and D are equivalent with respect to $\mathcal{T}$, if $C^{\mathfrak{I}} = D^{\mathfrak{I}}$ for all models $\mathfrak{I}$ of $\mathcal{T}$. We write $\mathcal{T} \models C \equiv D$. Property equivalence is defined analogously.
- *Disjointness* Two classes C and D are disjoint with respect to $\mathcal{T}$, if $C^{\mathfrak{I}} \cap D^{\mathfrak{I}} = \emptyset$ for all models $\mathfrak{I}$ of $\mathcal{T}$.

Example 2.4.3 *The class* PERSON *subsumes* FATHER *in the TBox presented in Table 2.5 on page 35. All classes in this TBox are satisfiable.*

2.4.3.2. Reduction to other inferences

For Description Logics without full negation, e.g. $\mathcal{AL}$, all inferences can be reduced to subsumption. For example, a class C is unsatisfiable iff C is subsumed by $\bot$. If, a Description Logic offers both intersection and full complement, satisfiability becomes the key inference of terminologies, since all other inferences can be reduced to satisfiability (cf. (Smolka, 1988)). Consequently, algorithms for checking satisfiability are sufficient to obtain decision procedures for any of the four inferences we have discussed. Moreover, this observation gave rise to the research on specialized tableau calculi which are used in the current generation of DL systems.

Elimination of the TBox If a TBox is acyclic, we can eliminate the TBox by expanding[13] all class descriptions using the equivalences and inclusions stated in the TBox. Therefore, it suffices to consider (expanded) class descriptions with respect to the inference problems presented above.

During the design phase of the TBox, the above mentioned inferences are typically used to determine whether all classes are satisfiable and that expected subsumption relationships hold.

2.4.3.3. ABox

Reasoning tasks usually come into play after the design of the TBox, e.g. all classes are typically satisfiable. As the ABox contains two kinds of assertions, viz. class assertions $C(i)$ and property assertions $R(a, b)$, reasoning tasks typically considered for ABoxes are the following:

- *Consistency* An ABox $\mathcal{A}$ is consistent with respect to a TBox $\mathcal{T}$,[14] if there is an interpretation $\mathfrak{I}$ that is a model of both $\mathcal{A}$ and $\mathcal{T}$. We say $\mathcal{A}$ is consistent, if it is consistent with respect to the empty TBox.
- *Instance checking* An assertion α is entailed by $\mathcal{A}$ ($\mathcal{A} \models \alpha$), if every interpretation $\mathfrak{I}$ that satisfies $\mathcal{A}$ also satisfies α. A common reduction for $\alpha = C(a)$ is to check whether $\mathcal{A} \cup \{\neg C(a)\}$ is inconsistent. (Schaerf, 1994) has shown that

[13] (Nebel, 1990) shows that the expansion can be exponential in the size of the original class description.

[14] If $\mathcal{T}$ is eliminated via expansion, the expansion must also be applied to $\mathcal{A}$.

ABox consistency can be reduced to class consistency, if a language possesses the $\mathcal{O}$ constructor. Otherwise instance checking is usually harder than class satisfiability (Donini et al., 1994).

- *Retrieval problem* Given an ABox $\mathcal{A}$ and a class C, find all individuals a such that $\mathcal{A} \models C(a)$. Dually we can find all named classes C for an individual a for which $\mathcal{A} \models C(a)$.
- *Property fillers* Given a property R and an individual i with respect to a TBox T and a ABox A, retrieve all individuals x which are related with i via R, viz. $\{x|(\mathcal{T}, \mathcal{A}) \models R(i, x)\}$. Similarly we can retrieve the set of all named properties R between two individuals i and j, ask whether the pair (i, j) is a filler of P or ask for all pairs (i, j) that are a filler of P.

Typically, DL systems give support for the stated TBox problems, however only one system, namely RACER (Haarslev & Moller, 2001), currently supports ABoxes and the above-mentioned inference problems.

2.4.4. Complexity

Naturally, the question arises how difficult it is to deal with the above reasoning problems. Traditionally, the complexity of reasoning has been one of the major issues in the development of Description Logics. While studies about the complexity of reasoning problems initially were focused to polynomial-time versus intractable (Brachman & Levesque, 1984), the focus nowadays has shifted to very expressive logics such as $\mathcal{SHIQ}$ whose reasoning problems are EXPTIME-hard or worse.

	no OR source	OR source
no AND source	$\mathcal{AL}$, $\mathcal{ALN}$ PTIME	$\mathcal{ALU}$, $\mathcal{ALUN}$ NP-complete
AND source	$\mathcal{ALE}$ coNP-complete	$\mathcal{ALC}$, $\mathcal{ALCN}$, $\mathcal{ALEN}$ (Hemaspaandra, 1999) PSPACE complete

Table 2.6.: Complexity of Satisfiability for $\mathcal{AL}$ languages w/o TBox

One can see that exponential-time behavior of some Description Logics is due to two independent origins: an AND source which corresponds to the size of a single model candidate and an OR source which is constituted by the number of model candidates that have to be checked. An example OR source is disjunction, where we can alternatively assign an element of a model to several classes and we have to check every alternative. No OR source therefore means that we can check the

validity of a single model. However, if an AND source such as the existential restriction is present, we may have to expand the model with new elements. Table 2.6 on page 40 classifies the several Description Logics of the $\mathcal{AL}$ family with respect to both sources. The complexity of the $\mathcal{S}$ family of languages that are relevant for the Semantic Web is listed in Table 2.7 on page 41.[15]

Description Logic	Complexity
$\mathcal{S}$	PSPACE-complete[16]
$\mathcal{SI}$	PSPACE-complete[17]
$\mathcal{SH}$	EXPTIME-complete
$\mathcal{SHIF}$	EXPTIME-complete
$\mathcal{SHIQ}$	EXPTIME-complete
$\mathcal{SHIOQ}$	NEXPTIME-hard

Table 2.7.: Complexity of Satisfiability for $\mathcal{S}$ languages

The presence of a cyclic TBox can increase the complexity of reasoning problems. Even for $\mathcal{AL}$, presence of a general TBox leads to EXPTIME-hardness (Nebel, 1990), which also effects Description Logics like $\mathcal{SH}$, which allow the internalization of general inclusion axioms. An extension with datatypes also effects the complexity. Given that we distinguish datatype properties, satisfiability is decidable if the inference problems for the concrete domain are decidable (Horrocks & Sattler, 2001).

Complexity results also give motivation why no other property constructors have been presented here. For example, adding role composition to $\mathcal{AL}$ already leads to undecidability of the Description Logic (Baader et al., 2003)[Section 5.9].

2.4.5. Evaluation Strategies

Early DL Systems who dealt with simple logics without full complement, e.g. $\mathcal{AL}$, used structural subsumption algorithms to deal with the subsumption problem (and consequently all other TBox problems). These subsumption algorithms operate on normalized class definitions and compare their syntactic structure. While these algorithms are sound and complete for simple languages and also show very efficient performance in practise, they cannot be used for expressive languages, which provide either disjunction or full existential restrictions (and consequently the full complement).

[15] Please note that these logics disallow the combination of transitive properties and cardinality restrictions, otherwise the logics are undecidable.

[16] Without TBox.

[16] Without TBox.

Modern systems such as RACER (Haarslev & Moller, 2001) use algorithms, which can be understood as specializations of FOL tableaux calculi. Again, these algorithms operate on the *negation normal form* (*NNF*)[17] of a class definition and then try to systematically construct a model for a class, viz. $C^{\mathfrak{I}} \neq \emptyset$.[18] The algorithm generates individuals and imposes constraints on it. If an individual can be found that satisfies all constraints, we have found an interpretation for the class. The constraint checking is done expanding the tableaux with tableau-expansion rules. Since these rules can be non-deterministic several expansions can exist, which are the explanation for many of the above mentioned complexity results. The modularity of tableau-expansion rules is beneficial for incorporating new constructors into a Description Logic, since we only need one further rule for each constructor. Hence, sound and complete tableau-based algorithms are known for all Description Logics presented here.

In order to make tableau algorithms usable (in spite of the discouraging tractability results) several optimizations are usually implemented, which lead to "empirical" tractability of realistic TBox problems even for expressive languages like $\mathcal{SHIQ}$. However no known "practical" algorithms for $\mathcal{SHIOQ}$ (D) exists. Nor do any "practical" algorithms exist for ABox problems in expressive Description Logics.

Notably, a third approach for the evaluation of Description Logics mainly stems from complexity analysis and is based on a reduction to automata, which operate on infinite trees. A class is satisfiable iff. the language accepted by an automaton is not empty (Calvanese et al., 2002)[Section 5.3].

[17] A NNF has the characteristic that negations only occur in front of atomic class names.
[18] This means that there must exist an individual in $\Delta^{\mathfrak{I}}$ that is an element of $C^{\mathfrak{I}}$.

3. The Semantic Web

Der Mensch besitzt die Fähigkeit Sprachen zu bauen,
womit sich jeder Sinn ausdrücken lässt,
ohne eine Ahnung davon zu haben,
wie und was jedes Wort bedeutet.
Ludwig Wittgenstein,
Tractatus Logico-philosophicus,
4.002

This chapter presents the architecture and components of the Semantic Web (Berners-Lee, 1999) whose aim is to increase machine support for the interpretation and integration of information on the World Wide Web (WWW). Since the Semantic Web is still in genesis, this chapter can only describe the state in February 2004[1].

The layered architecture (cf. Figure 3.1 on page 45), which was envisioned for the Semantic Web by its inventor Tim-Berners Lee (Berners-Lee, 2000b), will guide us through the chapter. Section 3.1 sketches the fundamental ideas behind the Semantic Web and introduces the layered architecture. Section 3.2 on page 46 recapitulates the syntax layer and briefly introduces the Extensible Markup Language (XML), the accompanying languages for defining schemas and discusses the inadequacy of XML as a semantic foundation of the Semantic Web. Section 3.3 on page 49 presents the data layer and describes the Resource Description Framework (RDF), which is intended to be the unifying data model for all Semantic Web data purposes. We discuss the proposed semantics for RDF data and the associated vocabulary definition language RDF Schema (RDFS) and present an (incomplete) axiomatic formalization of RDF in Datalog. Section 3.4 on page 60 presents the ontology vocabulary layer and introduces the Web Ontology Language (OWL) and its subsets OWL Lite and OWL DL. We relate the language to the Description Logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ presented in Section 2.4 on page 31 and discuss limitations of the language. Section 3.5 on page 70 presents current proposals for rules on the Semantic Web and discusses their relation with the other layers presented before.

[1] In which both RDF and OWL reached W3C recommendation status

3.1. Introduction

The term Semantic Web, which was coined by the inventor of the web, Sir Tim Berners-Lee, in (Berners-Lee, 1999), stands for the idea of a future Web which aims to increase machine support for the interpretation and integration of information on the World Wide Web (WWW).

The World Wide Web Consortium (W3C), which is the standardization body responsible for the Web, defines the term Semantic Web in its "Semantic Web Activity Statement" (W3C, 2001) as follows:

Definition 3.1.1 (Semantic Web) *"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." (W3C, 2001)*

The need for a Semantic Web has been motivated by the fact that "the mix of content on the web has been shifting from exclusively human-oriented content to more and more data content." (W3C, 2001). Hence, techniques for "more effective discovery, automation, integration, and reuse across various applications [are needed] for the web to reach its full potential [...]" (W3C, 2001).

The Semantic Web provides a solution by having data defined and linked in a way such that these techniques can be provided on top. Eventually, the Semantic Web should provide an "[...] universally accessible platform that allows data to be shared and processed by automated tools as well as by people" (W3C, 2001).

According to Sir Tim Berners-Lee (Berners-Lee, 2000b) the following requirements must be fulfilled by a Semantic Web:

1. providing a common syntax for machine understandable statements,
2. establishing common vocabularies,
3. agreeing on a logical language,
4. using the language for exchanging proofs.

At the same time Berners-Lee suggested that these requirements depend on each other and invented a layered, functional architecture (cf. Figure 3.1 on page 45 from (Berners-Lee, 2000b)), aka. "Layer Cake", that addresses the requirements and other design issues formulated for the Semantic Web in (Berners-Lee, 1998). According to this architecture, the Semantic Web is expected to be built in an incremental fashion and each layer alone is able to provide added value by providing unique functionality.

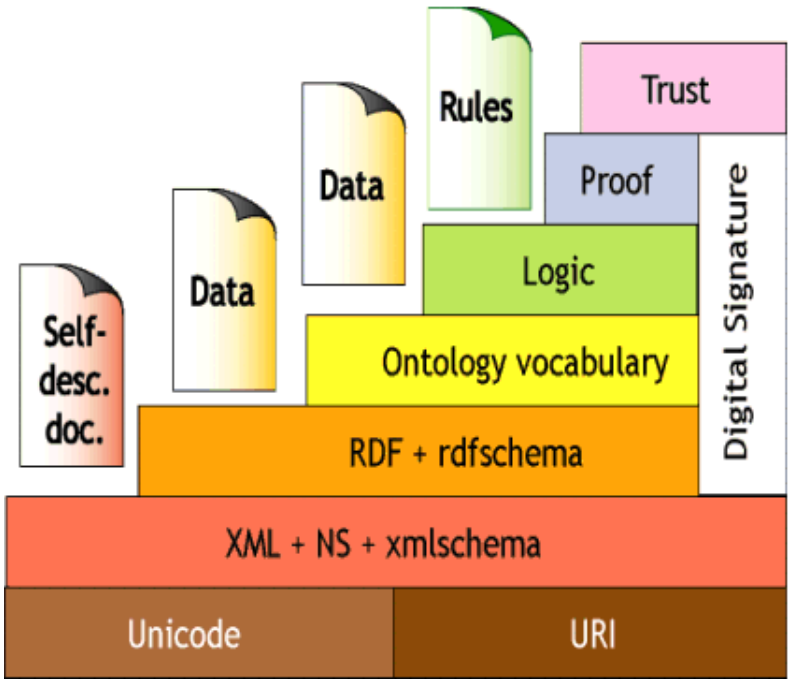


Figure 3.1.: Semantic Web Layer Cake (Berners-Lee, 2000b)

The Semantic Web architecture grounds itself on available standards for refering to entities, viz. *Uniform resource identifiers (URIs)* (Berners-Lee et al., 1998)[2], and encoding of character symbols, i.e. *Unicode* (The Unicode Consortium, 2003), and reuses existing Web technologies like the *Extensible Markup Language (XML)* (Bray et al., 2000) for syntactic purposes. The core layers of the architecture, viz. XML, RDF, Ontology and Logic will be the subject of the subsequent sections on Syntax (Section 3.2 on page 46), Data (Section 3.3 on page 49), Semantics (Section 3.4 on page 60) and Rules (Section 3.5 on page 70). The top layers providing *Proof* and *Trust* are starting to be addressed by research today.[3] Consequently, we cannot give an explicit account of these layers but only describe their intention. According to (Berners-Lee, 1998), the ability to check the validity of statements made in the (Semantic) Web is important. Therefore the creators of statements should be able to provide a *proof* of correctness of the statement which is verifiable by a machine.

[2] *URL (uniform resource locator)* refers to a locatable URI, e.g. an `http://...` address. It is often used as a synonym, although strictly speaking URLs are a subclass of URIs, see `http://www.w3.org/Addressing`.

[3] See (Richardson et al., 2003) for a first approach to *Trust* and (McGuiness & da Silva, 2003) for an approach to *Proof*.

At this level, it is not required that the machine that reads the statements finds the proof itself, it 'just' has to check whether the proof provided by the creator is feasible enough to *trust* the provided statements.

3.2. Syntax Layer

This section introduces the syntactic layer of the Semantic Web, where each collection of syntactic entities is constituted by documents that are encoded in the Extensible Markup Language (XML) (Bray et al., 2000). XML provides a standard way to encode documents and is surrounded by a family of specifications. For the Semantic Web context, two of these specifications are important. Firstly, the XML Schema standard (Thompson et al., 2001), which is primarily a mechanism to define grammars for legal XML documents. XML Schema additionally provides a number of predefined datatype definitions and rules to define and derive new datatypes. Secondly, the XML Namespaces (XMLNS) specification (Bray et al., 1999), which allows to use multiple heterogeneous vocabularies within a single document. The following subsections describe these specifications further.

3.2.1. Extensible Markup Language (XML)

XML (Bray et al., 2000) is a meta language for creating markup languages. Markup languages are usually intended to specify a syntactic encoding of documents allowing their exchange between applications. To this extent, XML defines a set of syntactic rules that allow to regard documents as labelled trees. Documents that obey the syntactic rules of XML are called *well-formed*.

Example 3.2.1 (XML Namespaces) *Namespace usage in an RDF/XML document:*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF xmlns="http://www.jsbach.org/bach#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
    <rdf:Description rdf:about="#BWV214">
        <rdf:type rdf:resource="#BirthdayCantata"/>
        <hasTitle>
            Tönet, ihr Pauken, erschallet, Trompeten !
        </hasTitle>
        <hasLyrics rdf:resource="#Lyrics-BWV214"/>
```

```
        <inDeferenceTo rdf:resource="#Friedrich-von-Sachsen"/>
    </rdf:Description>
</rdf:RDF>
```

The well-formed XML document in Example 3.2.1 creates a balanced tree of nested sets of named elements. Each element (e.g. `rdf:Description`) is started and ended by so-called tags, and might itself contain other elements (e.g. `hasTitle`). The name of the starting and closing tag of each element must be the same[4] (e.g. `<hasTitle>` and `</hasTitle>`) and is used to label the element. Each element can include several attribute-value pairs (e.g. `name="rdf:about"`) which are stated together with the starting tag.

3.2.2. XML Namespaces

Different XML applications can easily declare the same vocabulary for attribute or element names. The XML Namespace specification (Bray et al., 1999) has been devised to distinguish the names of different applications. This is achieved by prefixing each name with a *namespace*, which must be declared as an attribute in one of the superelements (preferably the root element). As we can see in Example 3.2.1 on page 46, each namespace declaration assigns an URI fragment to a shortcut. When the XML document is parsed, all names are expanded with the namespace URI and can be uniquely identified by the respective XML application.

3.2.3. XML Schema Languages

XML itself is based on the simple idea of representing documents as trees. However, many applications can only make use of special trees, viz. documents that are *valid* by conforming to a predefined structure and vocabulary. Tree grammars are used to restrict the structure of a document. Such tree grammars can be defined using two W3C specifications, Document Type Definition (DTD) (Bray et al., 2000) and XML Schema (Thompson et al., 2001). XML Schema is more powerful than DTDs[5], since it supports the derivation of element types (similar to subclassing in object-oriented languages), permits 'all' groups and nested definitions and

[4] If an element contains no subelements the starting and closing tag must fall together, e.g. `inDeferenceTo` in Example 3.2.1 on page 46.

[5] XML Schema is also more complex than DTDs, requiring ten times more pages for its description compared to XML 1.0 (which included DTDs). Unfortunately this complexity also leads to the situation that no complete formal specification of the XML Schema language exists (cf. (Brown et al., 2001; Simeon & Wadler, 2003) for first approaches).

provides atomic data types (such as integers, floating point, dates, etc.) in addition to character data. Currently, the primary role of XML Schemas in the Semantic Web is the provision of data types (cf. Figure 3.2 on page 49), which can be used to type element content and attributes. Example 3.2.2 on page 48 illustrates how the datatype "BWVEntry" can be defined using XML Schema.

Example 3.2.2 (Complex type definitions) *An entry in the Bach Werke Verzeichnis (BWV) is composed of a single number, written for at least one instrument and might have a creation year:*

```
<xsd:complexType name="BWVEntry">
    <xsd:sequence>
        <xsd:element name="number" type="xsd:int"
                     minOccurs="1" maxOccurs="1" />
        <xsd:element name="forInstrument" type="xsd:string"
                     minOccurs="1" maxOccurs="unbounded" />
        <xsd:element name="creationYear" type="xsd:gYear"
                     minOccurs="0" maxOccurs="1" />
    </xsd:sequence>
</xsd:complexType>
```

3.2.4. Discussion

It is important to note that DTD and XML Schema only specify syntactic and structural conventions for exchange of documents between (a set of) applications. Therefore, they were not designed to provide a semantic description of the domain in which the applications operate (Erdmann, 2001). This semantic description is outside the realm of the XML specification and is achieved by the upper layers of the Semantic Web layer cake.

Nowadays, many people use XML not only for document exchange purposes but also for data exchange where one can consider XML as a data model that allows to express non-first normal-form (NF2) relations (Abiteboul & Bidoit, 1986), yielding a very expressive data model. However, due to this expressivity and the polymorphism of possible syntactic encodings, the exchange of data where applications do not have an a-priori understanding of the data scheme, is limited (Tolle, 2000)[Section 2.4.3.3]. For example, data can be similarly encoded in attribute values or element contents. Additionally, it is not clear what the nesting of elements means. In many cases, it encodes the individual (possibly multi-valued) attributes of some information object, in other cases it encodes information about set membership.

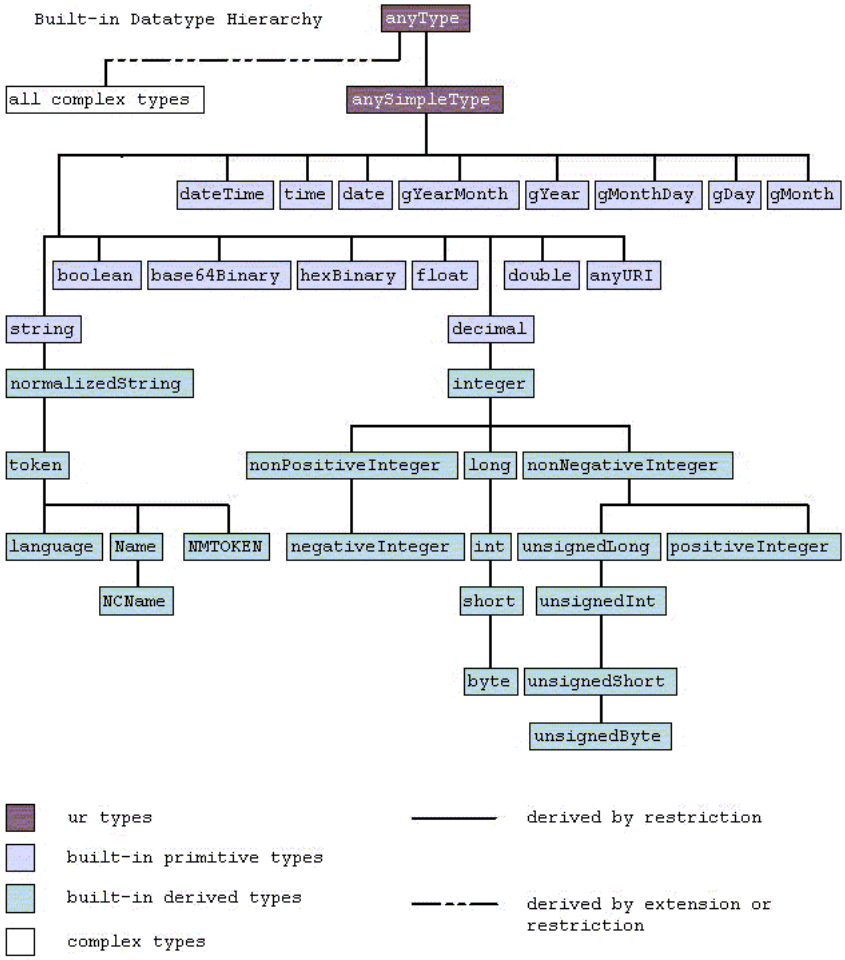


Figure 3.2.: Hierarchy of RDF-compatible XML Schema datatypes

3.3. Data Layer

3.3.1. Resource Description Framework (RDF)

The Resource Description Framework (RDF) (Klyne & Carroll, 2003) tries to improve data interoperability on the Web by specializing the XML data model. Our exposition of RDF excludes many aspects of the specification. For a full account, the interested reader may refer to the RDF suite of specifications (Beckett, 2003; Grant & Beckett, 2003; Manola & Miller, 2003; Klyne & Carroll, 2003; Hayes, 2003; Brickley & Guha, 2003).

3.3.2. Data Model

In essence, RDF allows to syntactically encode labeled directed graphs using a dedicated XML syntax. In RDF, the term graph is used synonymously with the term model.[6]

Vertex types In an RDF graph several types of vertices are distinguished:

- *Resource:* Resources are vertices which represent object identifiers. Resources are usually represented by an URI.
- *Literal:* Literals are vertices which denote data values. Each literal can be associated with an XML Schema datatype.
- *Property:* Properties are those resources, which are used as labels of graph edges.

Definition 3.3.1 (RDF graph) *Let L, R be the disjoint sets of literals and resources and $P \subset R$ be the set of properties. An RDF graph is a directed graph $G = (V, E)$ of disjoint sets of vertices $V = L \cup R$ and edges E together with three surjective functions*

- $subject : E \rightarrow R$*, which maps an edge to its initial vertex;*
- $object : E \rightarrow V$*, which maps an edge to its end vertex;*
- $property : E \rightarrow P$*, which associates an edge with its label;*

Remark 3.3.1 *In the following we will use the function name to denote the value of the function when applied to an edge $e \in E$.*

Statement The term *Statement* [7] is used in RDF to denote a graph edge $e \in E$ together with its property $property(e)$, subject $subject(e)$ and object $object(e)$.

Remark 3.3.2 *In the following, we will use the syntactic form $p(s, o)$ to express an edge e whose label $property(e)$ is p and $subject(e)$ is s and $object(e)$ is o, given that we are not interested in e itself but only in the values of the functions.*

Due to the atomicity of statements, individual information objects have to be represented in RDF using sets of statements having the same subject resource. Hence, object identity that is given via the uniform resource identifier (URI) is central in RDF data modeling.

[6] The reader may note that this has nothing to do with the concept of a model in model theory.
[7] Often the term "*Triple* " is used synonymously to statement.

3.3.3. RDF Syntax

People use several syntaxes for RDF. The main syntax, however, is the XML based syntax of RDF (RDF/XML). Example 3.3.1 on page 51 demonstrates how edges of an RDF graph can be asserted in RDF/XML using the fundamental syntactical element `rdf:Description`.

Example 3.3.1 (RDF Descriptions) *The labeled edges* `title(BWV248,"Christmas Oratorio")` *and* `reuses(BWV248,BWV213)` *can be syntactically represented as follows:*

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description rdf:ID="BWV248">
    <reuses rdf:resource="#BWV213"/>
</rdf:Description>
<rdf:Description rdf:ID="BWV248">
    <hasTitle rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    Christmas Oratorio
    </hasTitle>
</rdf:Description>
</rdf:RDF>
```

The attribute `rdf:ID`[8] of a `rdf:Description` tag denotes the subject of a statement, the tag name of the nested subelement denotes the property of a statement, and the value of the `rdf:resource` attribute denotes the object of a statement. Alternatively, the object can also be the subelement content or the value of the `rdf:ID` attribute of the subsubelement (see Example 3.3.2 on page 52).

RDF provides numerous other ways to encode graphs, which are further described in (Beckett, 2003). For our context, two more syntactic variations are important. Firstly, the syntactic possibility to use the resource identifier of an object instead of `rdf:Description` to create statements of the fixed form `rdf:type(subject,object)`. For example, we can encode the statement `rdf:type(BWV248,Oratorio)` in RDF/XML as

```
<Oratorio rdf:ID="BWV248" />
```

Secondly, we can nest statements, which gives rise to resources without identifiers.

[8] Or `rdf:about`, with `rdf:about="#x"` ⇔ `rdf:ID="x"`.

3.3.3.1. Anonymous Resources

Anonymous resources[9] are created if resource identifiers are omitted in the assertion of an RDF/XML description[10]. Anonymous resources are used routinely in practise, when no identifier for a resource is known and can be logically understood as a existential quantification (Hayes, 2003)[Section 1.5], i.e. we only know that a resource must exist:

Example 3.3.2 (Nesting / Anonymous resources) *This example demonstrates the (anonymous) instantiation of an OWL restriction and shows nesting.*

```
<owl:Class rdf:ID="BirthdayCantata">
    <rdfs:subClassOf>
        <owl:Class rdf:ID="Homage"/>
        <owl:Class rdf:ID="Cantata"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty>
                <owl:ObjectProperty rdf:ID="forEvent"/>
            </owl:onProperty>
            <owl:someValuesFrom>
                <owl:Class rdf:ID="Birthday"/>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Anonymous resources are used extensively when RDF is used as a syntax carrier for another language. For example, the existential restriction constructor in a Description Logic can be represented by means of three statements (cf. Example 3.3.2), which are stating that some object is of type restriction and restricts a certain property (e.g. `forEvent`) to a certain class (e.g. `Birthday`). Obviously, the subject of these statements is only used to hold the statements together, therefore it is usually omitted.

Anonymous resources are also used to encode non-binary relations (Klyne & Carroll, 2003)[Section 3.5]. This shows, that the ternary statements offered by the RDF

[9] Often the term "blank node" is used synonymously to anonymous resource.

[10] Practically, the `rdf:about` attribute is omitted in the syntax and the nesting of XML is used to assign the object of a statement to the anonymous resource instantiated subelement.

data model are theoretically sufficient to represent n-ary relations. However, several limitations arise in practise.

Firstly, the intention of a collection of statements with an anonymous subject is not clear. One possible interpretation is that each statement encodes attribute value pairs of the anonymous object. Alternatively, we can understand statements as individual parameters of a n-ary relation. In this case we are typically only interested in the set of objects of a statement and not in the property URIs and the subject identifier, which are reduced to syntactical elements.

Secondly, RDF does not provide any syntactic means to enforce that all statements defining a n-ary predicate are there. For example, how would we interpret the restriction in Example 3.3.2 on page 52 if the onProperty statement is missing? In fact, this is one major source of problems when parsing a language like OWL, since all statements have to be present to understand the encoded constructs.

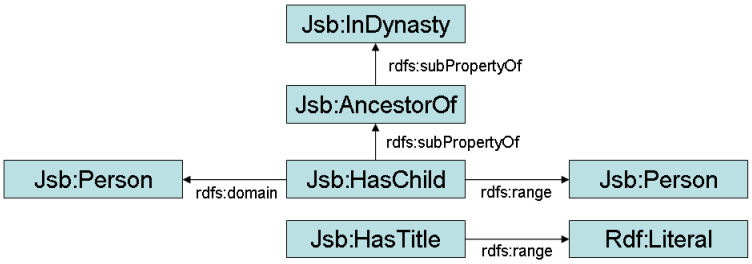


Figure 3.3.: RDFS Property Hierarchy

3.3.4. RDF Schema (RDFS)

The RDF vocabulary description language (RDF Schema [RDFS]) (Brickley & Guha, 2003) defines a simple modeling language on top of RDF. RDFS is intended to provide the primitives that are required to describe the vocabulary used in particular RDF models. This description is achieved by expressing set membership of objects in property and class extensions. Therefore RDFS uses classes, subsumption relationships on both classes and properties (cf. Figure 3.3 above), and global domain and range restrictions for properties as modeling primitives. When compared to typical object-oriented modeling languages, RDFS exposes a peculiar notion of object orientation:

1. RDFS treats properties as first class citizens, viz. they can exist independently of classes.

2. Global domain and range restrictions on properties are entailment rules, i.e. they assert that the subject (object) of a statement where the property occurs is a member of the classes stated in the domain (range) restriction. This departs from the usual constraint interpretation, which most object-oriented formalisms take for class attributes and associations.

3. Multiple domains and range restrictions can be defined on properties. If the domain or range of a property is not defined, the instantiation of its domain or range may occur to any resource-value pair, e.g. this applies to `Jsb:inDynasty` in Figure 3.3 on page 53.

4. RDFS allows cycles in its subsumption hierarchies. Cycles can be used to express the equivalence of classes or properties.

5. RDFS allows objects to instantiate multiple classes simultaneously, which is typically disallowed in object-oriented languages.

6. The RDFS language has a cyclical metamodel. The language elements itself are part of the vocabulary. For example, the resource `rdfs:Class` is an instance of itself (cf. Figure 3.4 below) and thereby always included in the extension of a class. Similarly `rdf:Resource` is an instance of `rdfs:Class`, while it subsumes `rdfs:Class`.

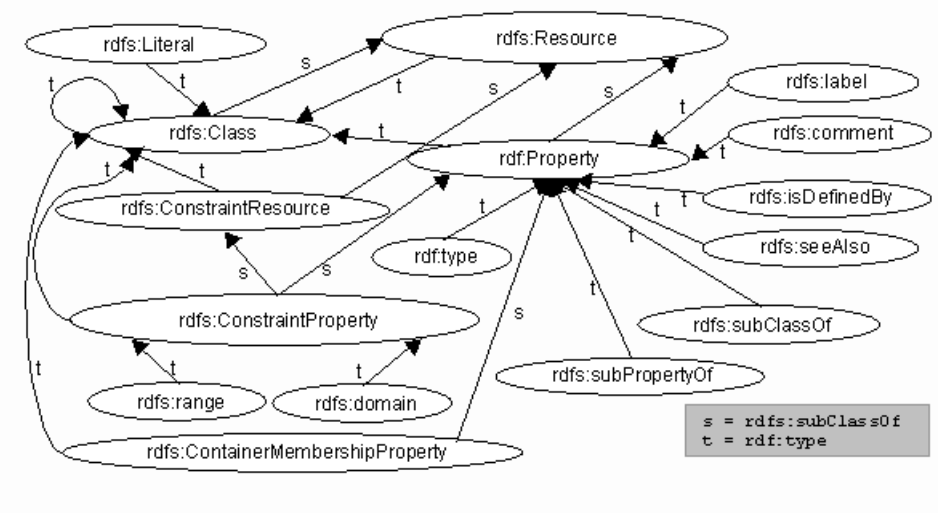


Figure 3.4.: Class Hierarchy for RDF Schema (Brickley & Guha, 1999)

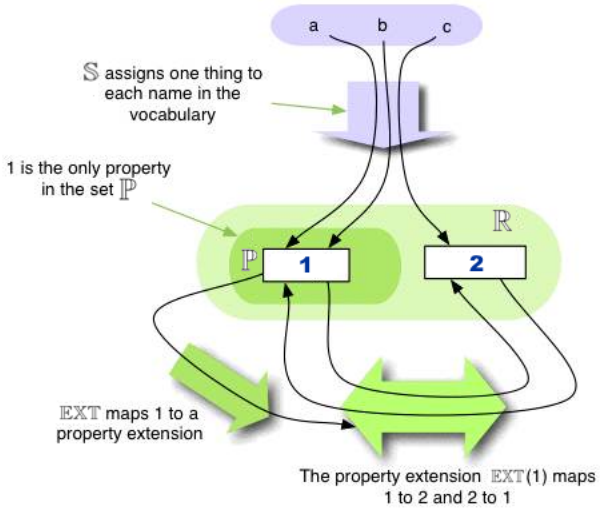


Figure 3.5.: RDF Model Theory: Interpretation

3.3.5. RDF Semantics

Unfortunately, the initial RDF specification (Lassila & Swick, 1999) did not provide any formal semantics for RDF. This lead to a series of alternative interpretations of RDF (c.f. (Broekstra et al., 2001; Nejdl et al., 2001; Conen & Klapsing, 2000; Tolle, 2000; Pan & Horrocks, 2001; Hayes, 2003; Guha & Hayes, 2003)). For example, (Nejdl et al., 2001; Conen & Klapsing, 2000; Tolle, 2000) interpreted domain and range restrictions in the initial RDFS specification (Brickley & Guha, 1999) in a constraint way, while (Pan & Horrocks, 2001; Hayes, 2003) interpret domain and ranges as entailment rules.

3.3.5.1. RDF Model Theory

The work on an official formal semantics of RDF (Hayes, 2003) has started three years after the first official specifications of RDF have appeared. It has now evolved into the semantics that is officially recommended by the W3C[11].

The RDF model theory uses a standard Tarski-style model theory to assign semantics to RDF graphs. This assignment of semantics is incomplete, since there are several aspects of meaning in RDF which are ignored by the model theory; for example it ignores all aspects of meaning encoded in particular URI forms (Berners-

[11]This is also suggested by the name change from RDF Model Theory (Hayes, 2001) to RDF Semantics (Hayes, 2003).

Lee et al., 1998) by treating all URI references as simple names. Moreover, some parts of the RDF and RDFS vocabularies are assigned less formal meaning than one might expect[12], e.g. RDF reification and containers (cf. (Hayes, 2003)[Section 3.2]).

Definition 3.3.2 (RDF Interpretation) *A simple interpretation $\mathfrak{I}$ of a set of vertices V in an RDF graph is defined by:*

- *A non-empty set $\mathbb{R}$ called the domain or universe of $\mathfrak{I}$;*
- *A set $\mathbb{P}$, called the set of properties of $\mathfrak{I}$;*
- *A mapping $\mathbb{EXT}$ from $\mathbb{P}$ into the powerset of $\mathbb{R} \times \mathbb{R}$, i.e. the set of sets of pairs of domain elements (x, y);*
- *A mapping $\mathbb{S}$ from URIs in R into $\mathbb{R} \cup \mathbb{P}$.*
- *A mapping $\mathbb{L}$ from typed literals in L into $\mathbb{R}$.*

The interpretation $\mathfrak{I}$ connects the set of nodes V (URIs) of the graph with the domain of the interpretation $\mathbb{R}$ using the functions $\mathbb{S}$ and $\mathbb{L}$. For example, $\mathbb{S}$ maps a to the domain element 1 in Figure 3.5 on page 55, viz. $\mathbb{S}(\texttt{a}) = 1$. Central to the RDF model theory is the extension function $\mathbb{EXT}$, whose role is to map a domain property into its extension, which is a set of pairs. In Figure 3.5 $\mathbb{EXT}$ maps 1 to $\mathbb{EXT}(1)$, which is a set of pairs $\{(1, 2), (2, 1)\}$.

Although classes have a distinguished role in RDFS, class primitives are not fundamental primitives in the RDF model theory. A system can detect all classes that have been declared in a particular RDF schema by filtering the extension of the central property `rdf:type`. Hence the extension of a class c could be defined as follows:

$$ext(c) = \{i | (i, c) \in \mathbb{EXT}(\mathbb{S}(\texttt{rdf:type}))\}$$

The denotation of an RDF graph G in the interpretation $\mathfrak{I}$ is given recursively by several rules, which extend the interpretation mapping $\mathfrak{I}$ from resources to graphs. For example, the interpretation of a statement $p(s, o)^{\mathfrak{I}}$ is only true, if $s, p, o \subseteq V$ and $p^{\mathfrak{I}} \in \mathbb{P}$ and $\langle s^{\mathfrak{I}}, o^{\mathfrak{I}} \rangle \in \mathbb{EXT}(p^{\mathfrak{I}})$, otherwise $p(s, o)^{\mathfrak{I}}$ will be false.

[12] "The omission of these conditions from the formal semantics is a design decision to accomodate variations in existing RDF usage and to make it easier to implement processes to check formal RDF entailment. For example, implementations may decide to use special procedural techniques to implement the RDF collection vocabulary." (Hayes, 2003)[Sec. 3.2 Para. 2].

3.3.5.2. Semantic Conditions

In order to get an RDF interpretation of an arbitrary graph, the resources V of that graph are interpreted together with the resources that are used to define the central RDF vocabulary itself. We will concentrate on the central RDF vocabulary R_{RDF}[13] for the purpose of our presentation. R_{RDF} is defined as follows:

$$R_{RDF} = \{\text{rdf:type, rdf:Property, rdf:XMLLiteral}\}$$

of RDF itself. Such an interpretation $\mathfrak{I}$ of $(V \cup R_{RDF})$ must satisfy *semantic conditions* that are stated for R_{RDF}.

Example 3.3.3 (RDF Semantic Condition) *The following condition defines $\mathbb{P}$ as the set of those elements in $\mathbb{R}$, which are in the extension of $rdf : type^{\mathfrak{I}}$ and associated with $rdf : Property^{\mathfrak{I}}$.*

$$x \in \mathbb{P} \text{ if and only if } \langle x, \texttt{rdf:Property}^{\mathfrak{I}} \rangle \in \mathbb{EXT}(\texttt{rdf:type}^{\mathfrak{I}})$$

Such subsets of the universe are central, i.e. are used frequently, in interpretations of RDFS.

In addition, the interpretation of an RDF graph must satisfy the *axiomatic triples* that define RDF itself. These statements can be considered to form a special graph

$$G_{RDF} = \{\text{rdf:type(rdf:type,rdf:Property), rdf:type(rdf:nil,rdf:List)} \ldots\}$$

Hence, an interpretation $\mathfrak{I}$ of a graph G must satisfy $(G \cup G_{RDF})$.

(Hayes, 2003) states further *semantic conditions* and *axiomatic triples* which an RDF interpretation has to fulfill. Together semantic conditions and axiomatic triples describe the semantics of RDF (Hayes, 2003)[Section 1 + 3] and RDFS (Hayes, 2003)[Section 4].

Interpretations are used for entailment, which is the main semantic relationship in RDF.

Definition 3.3.3 (RDF entailment) *An RDF graph G_1 entails an RDF graph G_2, written $G_1 \models G_2$, exactly when all interpretations $\mathfrak{I}$ satisfying G_1 also satisfy G_2.*

[13] (Hayes, 2003) use the abbreviation crdfV to denote this set.

3.3.5.3. Limitations

As we can see, the RDF model theory does not distinguish between RDFS language elements and other objects in the domain of discourse, since classes and properties are just objects in the domain of discourse. Moreover, properties like rdfs:subClassOf, that are typically used to define particular graphs, have a dual role (Broekstra et al., 2001; Nejdl et al., 2001) as properties that are used at the same time to define the language itself.

This dual role introduces self referentiality into the RDF Schema definition, which makes RDF Schema difficult to understand,[14] both for humans and applications. Both have to resolve the ambiguity which is presented by assembling multiple, conceptually different modeling primitives into one primitive. Consider for example an ontology editor, which has to distinguish whether the statement `rdf:type(subject,rdfs:Class)` defines a new language primitive "subject" or a new class "subject" in the ontology to come up with an appropriate visual representation of the statement.

Several other problems appear if an extension of the modeling language with more expressive constructs is envisioned (such as done for OWL):

- *Too few entailments:* (Patel-Schneider & Fensel, 2002) Too few entailments are derived by the RDF semantics for extensions like OWL, since all classes are also objects. For example, the assertion of `Daniel` being an instance of the class `(Student⊔Employee⊔European)` does not give rise to the entailment that `Daniel` is also an instance of `(Student⊔Employee)`.
- *Contradiction Classes:* (Patel-Schneider & Fensel, 2002) Contradiction classes are easily created if an extended language would allow to state cardinality constraints. For example, stating the cardinality constraint ≤ 0 `rdf:type` would make it impossible to determine class membership.
- *Size of the universe:* (Horrocks & Patel-Schneider, 2003) The model theory makes all syntactic elements of the language part of the domain. Hence, an extended language can easily constrain the size of the universe.

3.3.6. Axiomatic Semantics

An alternative way to specify the semantics of RDF is to give a translation from RDF into another formal logic with well defined semantics. This approach has been

[14] Consider for example the fact, that rdfs:Class is an instance of rdf:Resource and rdfs:Class is a sub-class of rdf:Resource. We can thereby deduce that rdf:Resource is an instance of its sub-class.

rdf1	t(P,type,Property) :- t(S,P,O).
rdfs2	t(S,type,C) :- t(P,domain,C), t(S,P,O).
rdfs3	t(O,type,C) :- t(P,range,C), t(S,P,O).
rdfs4a	t(S,type,Resource) :- t(S,P,O).
rdfs4b	t(O,type,Resource) :- t(S,P,O).
rdfs5a	t(P,subPropertyOf,R) :- t(Q,subPropertyOf,R), t(P,subPropertyOf,Q).
rdfs6	t(S,R,O) :- t(P,subPropertyOf,R), t(S,P,O).
rdfs7	t(C,type,Class) :- t(C,subClassOf,Resource).
rdfs8	t(A,subClassOf,C) :- t(B,subClassOf,C), t(A,subClassOf,B).
rdfs9	t(S,type,B) :- t(S,type,A), t(A,subClassOf,B).
rdfs11	t(X,subClassOf,Literal) :- t(X,type,Datatype).
rdfs12	t(Resource,subClassOf,Y) :- t(X,domain,Y), t(rdf:type,subPropertyOf,X).

Table 3.1.: Datalog Axiomatization of RDF(S) Constructors(Hayes, 2003)[Section 7]

suggested repeatedly with various alternative versions of the target logical language (e.g. (Conen & Klapsing, 2000; Fikes & McGuiness, 2001; Decker et al., 1998; Guha & Hayes, 2003; Roo, 2002)). For example, Table 3.1 on page 59 presents the core of the axiomatization used in the Euler system (in Datalog syntax). However, no proof of actual conformance of an axiomatization to the official model theoretic semantics has been given yet.

Nevertheless, axiomatic semantics have advantages for processing, since they render a direct implementation in systems and are more readable.[15] Notably, a pure axiomatization of the language does not suffice for RDF, since the model theory includes so-called "axiomatic triples", i.e. statements which are necessarily part of all possible RDF interpretations. An axiomatization must therefore be complemented by an initial set of facts that capture these axiomatic triples.

A translation also has to consider how syntactic elements are connected with the target logics. For the logics presented in Chapter 2 on page 13 this could be done by translating each RDF resource (with a URI) to a logical constant in the domain of discourse and translating each anonymous RDF resource into a distinct variable. As suggested in an earlier version of the RDF model theory (Hayes, 2001)[Section 3.1], each RDF statement can be translated into a clause using a single ternary predicate, say `t`. An RDF graph can then be translated into a single FOL formula that is the existential closure of the conjunction of all statements in the graph.

Alternatively, we can deal with anonymous resources using skolemization. Skolemization itself is entailment preserving for an RDF graph as the skolemization

[15] The interested reader is invited to compare the entailment rules with semantic conditions in (Hayes, 2003).

lemma of (Hayes, 2003) shows. A skolemized expression has the same entailments as the original expression provided that the original RDF graph did not contain the new skolem constants. If anonymous resources are removed in this way, RDF graphs can be translated into Datalog programs by considering each statement as a fact and augmenting the Datalog program with the rules that axiomatize the RDF semantics and additional facts capturing the axiomatic triples.

Any Datalog reasoner will then compute the minimal Herbrand model for the given Datalog program. As the Herbrand lemma of (Hayes, 2003) shows, this interpretation is a correct interpretation of the graph. Of course it is only one, namely the smallest possible, interpretation that satisfies the graph.

3.4. Ontology Layer

"Ontologies figure prominently in the emerging Semantic Web as a way of representing the semantics of documents and enabling the semantics to be used by web applications and intelligent agents."(Heflin et al., 2002)[Section 1.1]

Definition 3.4.1 (Ontology) *Ontologies are models that represent an abstraction of a domain in a formal way, such that several parties are able to agree on the abstraction and reuse the model in their own (Web) applications.*

The reader may note that the definition of ontology is not bound to a particular formalism but to the aspect of sharing a model. However, the latter aspect requires that ontologies are represented in some formal language, such that "detailed, accurate, consistent, sound and meaningful distinctions can be made" (Heflin et al., 2002)[Section 1.1]. As (Studer et al., 1998) pointed out, many formal knowledge representation mechanisms may play the role of such a modeling language in the Semantic Web context.

3.4.1. Web Ontology Language (OWL)

For example, the RDF vocabulary description language (RDF Schema) already provides a simple language for describing the vocabulary of RDF data. However, "more complex relationships between entities including: means to limit the properties of classes with respect to number and type, means to infer that items with various properties are members of a particular class, a well-defined model of property inheritance, and similar semantic extensions" (W3C, 2003) were identified to be required for a modeling language for the Semantic Web. To come forward with

such a language, several research projects and subsequently W3C standardization, chose Description Logics (cf. Section 2.4 on page 31) as a logical basis for an expressive ontology language. The recently proposed Web Ontology Language (OWL)[16] can be considered as a particular Description Logic.[17] The rest of this section will provide a summary of the OWL language. Interested readers may refer to the specifications (Smith et al., 2003; McGuinness & van Harmelen, 2003; Dean & Schreiber, 2003; Heflin et al., 2002; Patel-Schneider et al., 2003; Carroll & De Roo, 2003) for a full account.

3.4.2. Syntax

OWL is syntactically layered on RDF. Therefore, the official[18] syntax of OWL is the syntax of RDF. However, OWL extends RDF with additional vocabulary that can be interpreted as OWL ontologies when used to form particular RDF graphs.

Definition 3.4.2 (OWL Vocabulary) *Let the set R_{OWL} be the URIs of the OWL language primitives*

$$R_{OWL} = \{owl{:}Class, owl{:}Property, owl{:}Restriction, \ldots\}$$

Consequently, OWL ontologies can be encoded in normal RDF/XML documents and parsed into ordinary RDF graphs (cf. examples 3.2.1 on page 46 and 3.3.2 on page 52). The additional vocabulary defined by OWL suggests that OWL is a Description Logic. However, due to its syntactic encoding in RDF, it differs from the Description Logics presented in Section 2.4 on page 31 in several ways:

1. N-Ary language constructs such as conjunction ($\sqcap$) have to be encoded into several RDF statements (cf. Section 3.3.3.1 on page 52).

2. OWL graphs can contain circular syntactic structures which are not possible in Description Logics.

3. Due to the circular meta-model of RDF, classes and properties can be made instances of themselves. For example, it would be possible to assert the statement `rdf:type(owl:Class,owl:Class)`.

[16] The false acronym OWL was created by a consensus decision of the WebOnt working group.

[17] In fact, the standardization of OWL was based on previous DL-based ontology languages, i.e. DAML+OIL (Horrocks et al., 2001), and Ontology Inference Layer (OIL) (Fensel et al., 1999) that have been proposed by different research projects.

[18] There are several other proposals for a syntax of OWL, e.g. in plain text `http://owl.man.ac.uk/2003/concrete/latest/grammar.bnf`.

4. The identifiers used for classes, properties and individuals do not have to be disjoint.

These syntactic differences require that the OWL semantics departs from classical Description Logic semantics.

3.4.2.1. Language species

Several subsets of OWL were defined in the standard to accommodate various interest groups and allow to safely ignore language features that are not needed for certain applications.

Definition 3.4.3 (OWL Full) *OWL Full contains all constructors of the OWL language and allows the arbitrary combination of those constructors.*

The semantics of OWL Full are an (direct) extension of the RDF semantics. Unfortunately, this extension leads to various semantically surprising effects (cf. Section 3.3.5.3 on page 58) and yields an undecidable language. The latter fact follows directly from (Horrocks & Sattler, 2001), where the authors show that the combination of transitive properties and cardinality restrictions leads to undecidability. Hence, it is only possible to embed the language in conventional FOL in such a way that standard FOL provers can be used as sound reasoners, which might be incomplete due to the undecidability of FOL. Such a mapping to FOL is straightforward and has been implemented as part of the OWL API (Bechhofer et al., 2003b). For these reasons, we do not make use of the full OWL language in this dissertation.

Definition 3.4.4 (OWL DL) *OWL DL is a subset of OWL Full that makes several restrictions:*

1. *The OWL vocabulary can not be used as identifiers for classes, properties or individuals;*
2. *Class constructors with syntactic cycles are disallowed. For example, a restriction may not use itself such as in the OWL graph {rdf:type(x,owl:Restriction), owl:allValuesFrom(x,x)};*
3. *The sets of identifiers used for classes, properties, and individuals must be disjoint;*
4. *Cardinality restrictions may not be stated on properties which are transitive.*

From a processing perspective, several constraints are imposed on RDF graphs, such that certain edges are disallowed. For example, the third condition on the graph could be expressed as follows:

$$\forall e_1, e_2 \in E.((o(e_1) \neq o(e_2)) \leftarrow (p(e_1) = p(e_2) = \text{rdf:type}) \wedge$$
$$(s(e_1) = s(e_2)) \wedge (\{o(e_1), o(e_2)\} \cap V_{\text{CoreOWL}} \neq \emptyset))$$

where V_{CoreOWL} = {owl:Class, owl:ObjectProperty, owl:DatatypeProperty} and (s, p, o) are abbreviations for the functions $subject, property, object$.

Definition 3.4.5 (OWL Lite) *OWL Lite is the subset of OWL DL, which is created by imposing the following restrictions:*

1. *Disallows the usage of the following elements of the OWL DL vocabulary:*

   {*owl:intersectionOf, owl:unionOf, owl:complementOf,owl:oneOf, owl:hasValue*}

2. *Limits the values of stated cardinalities to 0 and 1;*
3. *Requires the usage of class identifiers in OWL restrictions;*

Unfortunately, it does not always suffice to simply disallow certain syntax constructs in order to simplify a language. If this is the case, a reasoner still has to deal with the syntactically impoverished but semantically expressive language. OWL Lite still allows to express `IntersectionOf`, `UnionOf` and `ComplementOf` by clever combination of other primitives. As an example, the following corollary shows how negation can be constructed.

Corollary 3.4.1 (Negation is in OWL Lite) *Negation is semantically available in OWL Lite.*

Proof: Let R be an object property. We define the classes A and NotA using primitives that are syntactically allowed in OWL Lite, as follows: $A \equiv \exists R.\top$ and $NotA \equiv \forall R.\bot$. $NotA^{\mathfrak{I}} = \Delta^{\mathfrak{I}} \backslash A^{\mathfrak{I}}$ follows for all interpretations $\mathfrak{I}$.

□

Additionally, several of the primitives offered by OWL can be considered as syntactic shortcut for other primitives (cf. Table 3.2 on page 64).

OWL Syntactic Shortcuts	
DisjointClasses($d_1 d_2$)	InverseFunctionalProperty(P):
$d_1 \sqsubseteq \neg d_2$	$\top \sqsubseteq \leq 1\, P^-$
SymmetricProperty(P):	domain(P, C):
$P \equiv P^-$	$\top \sqsubseteq \forall P^-.C$
FunctionalProperty(P):	range(P, C):
$\top \sqsubseteq \leq 1\, P$	$\top \sqsubseteq \forall P.C$

Table 3.2.: Syntactic Shortcuts

3.4.2.2. Abstract Syntax

An abstract syntax (Patel-Schneider et al., 2003)[19] has been developed for OWL DL, which acknowledges the OWL DL restrictions and is similar to the syntax used in many Description Logic reasoners. This compact representation will be used in the later parts of the thesis. The main class constructors[20] of OWL DL are captured in Table 3.3 on page 65, whereas Table 3.4 on page 67 describes the axioms and Table 3.5 on page 68 the possible assertions available in OWL DL.

3.4.3. Semantics

The semantics of OWL (Patel-Schneider et al., 2003) assigns meaning to all variants of OWL and is given using a standard model theory. However, the model theory is very complex to be understood even by skilled readers.[21] This is due to the integration with the RDF Semantics.

On the other hand, the semantics for OWL DL is fairly standard wrt. Description Logics. The domain of an OWL interpretation is a set whose elements can be divided into abstract objects ($\Delta^{\mathfrak{I}}$), and datatype values (Δ_D). Datatypes in OWL are derived from a subset of the built-in XML Schema datatypes (cf. Figure 3.2 on page 49). However, we will not consider the details of datatypes, here.

An interpretation in the OWL DL semantics is officially a four-tuple consisting of the domain $\Delta^{\mathfrak{I}} + \Delta_D$ and separate mappings for class identifiers C, property identifiers P, and individual identifiers I.[22]

[19] The abstract syntax follows the syntactic approach that has been taken for OIL (Fensel et al., 2001).

[20] We ignore annotations and deprecation, which allow to state information about classes and properties that is semantically uninterpreted and expand the grammar for the abstract syntax in our presentation such that the corresponding Description Logic primitives are directly visible.

[21] This is in part due to a myriad of unintuitive names which are used instead of the symbols used in papers and textbooks.

[22] The reader may note that these mappings are usually combined into a binary tuple in classical

An OWL DL ontology O is satisfied by an interpretation $\mathfrak{I}$ in the classical Description Logic sense. However, the main semantic relationship in OWL DL is not satisfiability but entailment - a relationship between pairs of OWL ontologies. This is similar to RDF. Hence, OWL inference is defined in terms of ontology entailment rather than ontology satisfiability such as defined for classical Description Logics (cf. Section 2.4.3 on page 38).

Definition 3.4.6 (OWL Entailment) *An ontology O_1 entails an ontology O_2, written $O_1 \models O_2$, exactly when all interpretations that satisfy O_1 also satisfy O_2.*

Entailment is not among the standard description logic reasoning problems, such as knowledge base and concept satisfiability (cf. Section 2.4.3 on page 38). Fortunately for our purpose, however, (Horrocks & Patel-Schneider, 2003) show that OWL DL and OWL Lite entailment can be reduced to checking Description Logic (un)satisfiability in the $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ (for OWL DL) and $\mathcal{SHIF}(\mathbf{D})$ (OWL Lite) Description Logics (cf. Table 2.2 on page 32 for a description of their constructors). For this purpose, they develop syntactic encodings of OWL into DLs that are similar to (Decker et al., 2000), who show how an OIL ontology can be encoded into a $\mathcal{SHIQ}$ TBox.

Constructor	Translation ρ
Classes	
A	A
`Thing`	$\top$
`Nothing`	$\bot$
`IntersectionOf` $(C_1 \dots C_n)$	$\rho[C_1] \sqcap \dots \sqcap \rho[C_n]$
`UnionOf` $(C_1 \dots C_n)$	$\rho[C_1] \sqcup \dots \sqcup \rho[C_n]$
`ComplementOf` (C)	$\neg\rho[C]$
`OneOf` $(o_1 \dots o_n)$	$\{o_1, \dots, o_n\}$
Object Property Restrictions	
`Restriction` $(R$	
`allValuesFrom` (C)	$\forall R.\rho[C]$
`someValuesFrom` (C)	$\exists R.\rho[C]$
`minCardinality` (l)	$\geq l\, R$
`maxCardinality` (m)	$\leq m\, R$
`cardinality` (n)	$= n\, R$
`hasValue` (o)	$\forall R.\{o\}$
$)$	

Table 3.3.: OWL DL Class Constructors (Abbr.)

Description Logics. This introduces no further problems since the two forms are equivalent.

3.4.3.1. Encoding OWL DL classes and axioms in $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$

Basically, each axiom and assertion in an OWL DL ontology is translated into one or more axioms in an $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ knowledge base. This is done by encoding each description into $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ class descriptions and encoding each OWL DL axiom into $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ axioms. The encoding is performed by a function ρ that is translates each OWL DL class constructor and each OWL DL axiom into $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ class definitions and axioms.

OWL class constructors (cf. Table 3.3 on page 65) can be arbitrarily nested. Each `Restriction` is stated with respect to a property R and may contain either one of the following components `allValuesFrom`, `someValuesFrom`, `minCardinality`, `maxCardinality` or `hasValue`.

OWL DL axioms are introduced using the constructors presented in Table 3.4 on page 67. The abstract syntax uses `Class` to introduce primitive classes. Classes can be either partially described, leading to a translation into a DL inclusion axioms, or completely described, leading to a translation into DL class equivalence axioms.

Example 3.4.1 (OWL DL Class Encoding) *For example, the OWL DL class definition*

$$\texttt{Class(}\ A\ \texttt{complete}\ C_1 \ldots C_n\ \texttt{)}$$

is translated into the $\mathcal{SHIN}\mathcal{O}(\boldsymbol{D})$ axiom

$$A \equiv C_1 \sqcap \ldots \sqcap C_n$$

by applying ρ as defined in Table 3.4 on page 67.

OWL DL Assertions (cf. Table 3.5 on page 68) are also encoded using the translation function ρ, but require a second function ϱ that encodes the different parts of individual assertions which are allowed in the abstract syntax. We also have to deal with the fact that anonymous individuals can be asserted.

3.4.3.2. Encoding OWL DL (anonymous) individuals in $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$

As mentioned before, OWL relies on RDF for the assertion of individuals. This can give rise to anonymous individuals in OWL if anonymous RDF resources are used for the assertion of OWL individuals. Likewise to RDF, such anonymous individuals can be understood as existentially quantified variables. We can apply the skolemization algorithm (cf. Figure 2.1 on page 19) to eliminate existential quantification. Since DLs can be understood as notational variants of function-free first

Axioms and Assertions	DL translation ρ
Class Axioms	
`Class` (A partial $D_1 \dots D_n$)	$\bigcup_{i\in[1,n]} A \sqsubseteq \rho[D_i]$
`Class` (A complete $D_1 \dots D_n$)	$\bigcup_{i\in[1,n]} A \equiv \rho[D_i]$
`EnumeratedClass` (A $o_1 \dots o_n$)	$A \equiv \{o_1 \dots o_n\}$
`DisjointClass` ($D_1 \dots D_n$)	$\bigcup_{1\le i<j\le n} \rho[D_i] \sqsubseteq \neg\rho[D_j]$
`EquivalentClasses` ($D_1 \dots D_n$)	$\bigcup_{i\in[1,n]} \rho[D_1] \equiv \rho[D_i]$
`SubClassOf` ($D_1 D_2$)	$\rho[D_1] \sqsubseteq \rho[D_2]$
Object Property Axioms	
`ObjectProperty` (P	
super(Q_1) ... super(Q_n)	$\bigcup_{i\in[1,n]} P \sqsubseteq \rho[Q_i]$
domain(D_1) ... domain(D_m)	$\bigcup_{i\in[1,n]} \top \sqsubseteq \forall P^-.\rho[D_i]$
range(D_1) ... range(D_l)	$\bigcup_{i\in[1,n]} \top \sqsubseteq \forall P.\rho[D_i]$
inverseOf(Q)	$P \equiv Q^-$
Functional	$\top \sqsubseteq \leq 1\, P$
InverseFunctional	$\top \sqsubseteq \leq 1\, P^-$
Symmetric	$P \equiv P^-$
Transitive	$P^+ \sqsubseteq P$
)	
`EquivalentProperties` ($P_1 \dots P_n$)	$\bigcup_{i\in[1,n]} P_1 \equiv P_i$
`SubPropertyOf` (P Q)	$P \sqsubseteq Q$
Individual Axioms	
`SameIndividuals` ($o_1 \dots o_n$)	$\bigcup_{i\in[1,n]} o_1 = o_i$
`DifferentIndividuals` ($o_1 \dots o_n$)	$\bigcup_{1\le i<j\le n} \neg(o_i = o_j)$

A is an atomic class name — P_i, Q_i are object property names
D_i are (complex) class descriptions — o_i are individual names

Table 3.4.: OWL DL Axioms

OWL DL Assertions	DL translation $\rho[constructor]$
Named Individual `Individual` ($o\ x_1 \ldots x_n$) Anonymous Individual `Individual` ($x_1 \ldots x_n$)	$\bigcup_{i\in[1,n]} \varrho[o, x_i]$ $\bigcup_{i\in[1,n]} \varrho[a, x_i]$ for a new unused skolem constant a
Assertion Part	DL Translation ϱ
ϱ[x,type(D)] ϱ[x,value(R o)] ϱ[x,value(U t)] ϱ[x,value(R `Individual` ($x_1 \ldots x_n$))]	$\rho[D](x)$ $R(x, o)$ $U(x, t)$ $R(x, a)$, ρ[Individual($a\ x_1 \ldots x_n$)] for a new unused skolem constant a

Table 3.5.: OWL DL Assertions

order formula (cf. Section 4.4.1 on page 88) it is sufficient to generate skolem constants. Following Lemma 2.2.1 on page 18, skolemization only preserves satisfiability. However, since all DL reasoning problems can be reduced to satisfiability checking (cf. Section 2.4.3 on page 38), this imposes no problem.

Example 3.4.2 (OWL DL Individual Encoding) *The OWL DL Individual assertion*

`Individual` *(type(C) value(R* `Individual` *(type(D))))*

is translated into the axioms $C(a)$, $R(a, b)$ and $D(b)$, where a, b are new skolem constants, by applying the translation function ρ as defined in Table 3.5 on page 68.

Corollary 3.4.2 *Skolemization is OWL entailment preserving.*

The corollary follows directly from (Horrocks & Patel-Schneider, 2003)[Theorem 2] and Lemma 2.2.1 on page 18.

Remark 3.4.1 *In our exposition we adapted the encodings of (Decker et al., 2000; Horrocks & Patel-Schneider, 2003) such that the (computationally expensive)* `OneOf` *class constructor is used as seldom as possible. We also encode property fillers differently[23] to (Horrocks & Patel-Schneider, 2003), i.e. we encode a single assertion $R(a, b)$ instead of introducing a new class B and encoding the two assertions $B(b)$ and $(\exists R.B)(a)$. (Horrocks et al., 2000) shows the equivalence of these encodings.*

[23] Since we avoid the usage of complex DL constructs as far as possible.

3.4.4. Complexity of OWL

As mentioned before, the OWL Full language is undecidable. Therefore complexity results are only of interest for OWL DL and OWL Lite. Complexity follows directly from the correspondence to Description Logics presented above. As the translation from OWL DL ontologies to $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ knowledge bases can be performed in polynomial time (Horrocks & Patel-Schneider, 2003), the results for the complexity of knowledge base satisfiability (cf. Section 2.4.4 on page 40) in $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ still hold, viz. OWL DL is NEXPTIME complete. Similarly, since the translation of OWL Lite axioms to $\mathcal{SHIF}(\mathbf{D})$ knowledge bases can be computed in polynomial time and results in a linear increase in size of the knowledge base, OWL Lite entailment has the same complexity as $\mathcal{SHIF}(\mathbf{D})$ knowledge base satisfiability, which is EXPTIME complete (cf. Section 2.4.4 on page 40).

3.4.5. Discussion

"Clearly, OWL is not the final word on ontology languages for the Semantic Web." (Horrocks et al., 2003b)[Section 8]. Since a number of useful features for a web ontology language were already identified in the OWL Requirements (Heflin et al., 2002) and never incorporated into the final language,[24] we briefly list some additional limitations that we could observe:

Ill-defined Layering The main limitations of OWL are due to the problems of layering the language on RDF (cf. Section 3.3.5.3 on page 58) and the politically defined layering within OWL. For the latter case, OWL Lite does not meet its initial anticipation, viz. is not the "easy" language that was initially motivated: "The goal of OWL Lite is to provide a language that is viewed by tool builders to be easy enough and useful enough to support. One expectation is that tools will facilitate the widespread adoption of OWL and thus OWL language designers should attempt to create a language that tool developers will flock to." (McGuinness & van Harmelen, 2002).

Monolithic Ontologies OWL Ontologies are monolithic in the sense that they can be separated into a set of ontologies, but have to be interpreted as one unified ontology. This is due to the fact that the only means for modularization is to include entire ontologies. This inclusion is fragile, since the included part is specified by location (which is usually different from the ontology identifier). Hence, one cannot import particular subsets of other ontologies.

[24] Requirements were rather weakened into objectives that could be safely ignored.

Datatyping Datatyping in OWL is still not worked out. This is due to the fact that the concrete domain Δ_D in $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ is not sorted. However, when reducing entailment to satisfiability, a datatype derived from the negation of a data value is needed. This negation would refer to the complete data domain, which is clearly unsatisfactory. For example, the negation of user-defined XML Schema datatype (> 5) that can be derived from the integer simple type would not be (≤ 5) as one would expect, but also include strings, floats etc. and all other elements of the data domain.

Tractability As we have seen in the previous section, the language exposes discouraging complexity results for its reasoning tasks. While several optimizations for the standard TBox problems are well known, which result in "empirical" tractability of realistic TBox problems, "empirical" tractability could not be achieved for ABox problems up till now. Moreover, no "practical" algorithm is currently known for any $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ reasoning problem.

3.5. The Logic Layer - Rules

Nowadays research usually considers the ontology and the logic levels together, as any semantic specification like ontologies has to be grounded in logic. Hence, the name "Logic" was not well chosen for this layer (cf. Figure 3.1 on page 45) and should rather be seen as a vertical layer that is orthogonal to the functional layers of the architecture.

As we have seen in the previous section, languages like OWL do not only specify a vocabulary and constrain the use of that vocabulary by restrictions, but also provide axioms, which allow to deduce new information from explicit information. However, OWL does currently not allow the definition of general rules over properties. For example, one cannot express property chaining in OWL.

Since there is, currently, no consensus on how a rule layer could look like, we only briefly describe the alternative proposals that have been made until now. We compare the individual proposals by means of a simple example rule (cf. Table 3.6 on page 71).

3.5.1. XML-based Approach - RuleML

This approach basically relies on specialized XML vocabularies to create various types of logic programs and their underlying knowledge bases. The most prominent representative today is RuleML (Boley et al., 2001), which stands in a line

Rule Language	Property Chaining example
Natural language	An uncle is the brother of the parent of a person.
Datalog	`isUncleOf(X,Y) :- isBrotherOf(X,Z), isParentOf(Z,Y).`

RuleML	`<imp>`
	`  <_head>`
	`  <atom>`
	`  <_opr>`
	`  <rel>isUncleOf</rel>`
	`  </_opr>`
	`  <var>X</var>`
	`  <var>Y</var>`
	`  </atom>`
	`  </_head>`
	`  <_body>`
	`  <and>`
	`  <atom>`
	`  <_opr>`
	`  <rel>isBrotherOf</rel>`
	`  </_opr>`
	`  <var>X</var>`
	`  <var>Z</var>`
	`  </atom>`
	`  <atom>`
	`  <_opr>`
	`  <rel>isParentOf</rel>`
	`  </_opr>`
	`  <var>Z</var>`
	`  <var>Y</var>`
	`  </atom>`
	`  </and>`
	`  </_body>`
	`</imp>`

Notation3 (N3)	`{ ?x :isBrotherOf ?z. ?z :isParentOf ?y. }`
	`  log:implies`
	`{ ?x :isUncleOf ?y. }.`

Triple	`FORALL model @rules(model) {`
	` FORALL X,Y X[isUncleOf->Y]<-X[isBrotherOf->Z]@model`
	` AND Z[isParent->Y]@model`
	`}`

Table 3.6.: Property Chaining in Rule Languages

of previous efforts like the Business Rules Markup Language (BRML), Agent-Object-Relationship Markup Language (AORML), Universal Rule Markup Language (URML) and Relational-Functional Markup Language (RFML).

Goal "The goal of the Rule Markup Initiative is to develop RuleML as the canonical Web language for rules using XML markup, formal semantics, and efficient implementations. RuleML covers the entire rule spectrum, from derivation rules to transformation rules to reaction rules [...]."[25]

Approach RuleML is developed in a modular specification and promises transformations from and to other rule standards/systems. A hierarchy of rule languages has been envisioned that builds more expressive languages from simpler languages, but eventually tries to provide all variants of rule languages, viz. everything from reaction rules, integrity constraints and derivation rules in logic programs. The modularized RuleML definition currently includes 12 sublanguages. The fourth row of Table 3.6 on page 71 defines an example rule in RuleML.

Limitations The main limitations of RuleML are due to the broad scope of the effort. It is still incomplete both syntactically and semantically. For example, a syntax (DTD) (RuleML v0.8) has only be designed for the Datalog subset. The lack of semantics is the main limitation of the effort, which describes its intention as follows "All sublanguages [...] correspond to well-known rule systems, where each sublanguage has a corresponding semantic (model- and proof-theoretic) characterization." (Boley et al., 2001). However, due to the multitude of different semantics used in different rule systems a clear, normative semantics should actually be the main requirement.

3.5.2. RDF-based Approaches - N3 and Triple

Other approaches to rules for the Semantic Web have chosen RDF graphs as knowledge bases. Every language provides a text-based syntax for the definition of rules. None of the systems provide built-in support for RDF semantics. They also usually do not automatically adhere to the RDF semantics.

Goals All approaches presented here share the goal of allowing to state rules on top of RDF graphs.

[25] `http://www.ruleml.org/` (Mission Statement).

Common approach All approaches operate on plain RDF graphs (with anonymous resources) and allow to specify a set of rules that provides some axiomatic form of entailment. Of course, this rule set could be an axiomatization of the RDF semantics (cf. Table 3.1 on page 59 for the incomplete RDF axiomatization used by the system Euler (Roo, 2002)).

3.5.2.1. Notation3 (N3)

Some systems like Euler (Roo, 2002) and CWM (Berners-Lee, 2000a) use Notation3 (N3) as syntax for rules. N3 was invented for the CWM reasoner (Berners-Lee, 2000a) as an alternative syntax of RDF that is easier to write since it is text-based and additionally allows to specify Horn-like rules.

The fifth row of Table 3.6 on page 71 defines an example rule in N3. The terms in braces { } are collections of (nested) statements, where `?x` denotes an all-quantified variable. N3 additionally allows a form of existential quantification to occur in the head of rules. All available N3 systems appear to deal with the existentials in the head of N3 rules by introducing new skolem constants. Therefore, the expressivity of N3 appears to be that of Datalog restricted to ternary predicates.

Limitations No formal account of the language has been given yet. Therefore the semantics and computational properties of N3 are not completely clear.

3.5.2.2. Triple

Triple reasons with RDF data in a frame-based syntax, which has been inspired by F-Logic (Kifer et al., 1995). It can operate on several RDF graphs similarly and distinguishes between graphs in a similar way to the Mediator Specification Language (MSL) (Garcia-Molina et al., 1997). Triple programs are based on Horn logic and are compiled into Prolog programs. Unlike F-Logic, Triple does not have a fixed semantics for object-oriented features like classes and inheritance. In order to support languages like OWL, which cannot be handled completely in Horn logic, Description Logic classifiers such as Racer (Haarslev & Moller, 2001) can be called as external systems. Hence, Triple can be regarded as a hybrid rule language.

The sixth row of Table 3.6 on page 71 defines the example rule in Triple. A set of Triple rules is associated with an identifier (`@rules` in the example). This identifier is used to identify the facts that have been derived by the respective set of rules. The rule set is instantiated on a parameterized set of source RDF graphs or other rule sets. These parameters (`@model` in the example) are then used to identify the set of

facts which should be used as ground data for the derivation. All derived data is automatically part of the rule set.

Limitations The limitations of Triple are fourfold. Firstly, Triple does not provide a rich set of built-in functions, such as available in most Logic Programming environments and also in many systems that support N3. Secondly, Triple is an undecidable language, since it allows unrestricted use of function symbols. Thirdly, the combination of Triple rule bases with Description Logic primitives yields an undecidable language, since the combination does not make restrictions such stated as other hybrid logics like CARIN (Levy & Rousset, 1996). Fourthly, a precise semantics of the triple language is still missing.

3.5.3. OWL-based Approaches

The first approach working towards a combination of OWL and rules was presented in (Grosof et al., 2003) and is detailed in the subsequent part of this thesis. Recently, a second proposal for an OWL rule language was made in (Horrocks et al., 2003a). Since this proposal was made during the time of writing this thesis, our discussion is very preliminary. The proposal suggests the extension of OWL axioms with horn clause rules.

Unlike the other rule languages presented in this chapter, an extension of the OWL model-theoretic semantics is given to provide a formal meaning for OWL ontologies including rules written in the proposed language. However, it is easy to see that the proposal yields an undecidable language since restrictions, such as stated for CARIN (Levy & Rousset, 1996), are not made and the proposed language can be used to simulate role value maps which yield undecidability (Schmidt-Schauß, 1989) when not further restricted. Given the undecidability result, it is questionable how one would attempt to practically implement the language in systems. Also, the motivations for other restrictions do not hold anymore. For example, one could well incorporate further undecidable features like function symbols and envision an extension to full FOL.

Part II.

Description Logic Programs

As we have seen in the previous chapter, Description Logic is expected to play a key role in ontology modeling for the Semantic Web applications. One of the main limitations of the ontology language, however, is its intractability (cf. sections 3.4 on page 60 and 2.4.5 on page 41). This part therefore develops Description Logic Programs (Grosof et al., 2003), a family of well-defined sub languages of OWL, which meets three goals.

Firstly, the language family can be supported by a large number of available logic databases and thereby immediately increases the number of systems, which can be utilized for the purposes of Semantic Web applications. For example, deductive databases and Logic Programming environments can reason with the new languages.

Secondly, the language family is tractable and can scale beyond toy examples with respect to ABox reasoning problems.

Thirdly, an extension of the language family to support rules in style of the approaches presented in Section 3.5 on page 70 is possible. A basic extension, namely a language for stating non-recursive rules, i.e. views, has been studied in (Volz et al., 2003e; Volz, 2003; Volz et al., 2002c; Volz et al., 2003d).

Our goals are achieved by translating an expressive subset of OWL into Logic Programs, which can be efficiently and tractably executed by numerous logic databases and Logic Programming environments.

In order to achieve our goals, however, we must

1. establish a meaning-preserving translation between DL and LP;
2. define a new DL language that can be translated into LP and explain how the typical reasoning and inferences available for DLs can be effected in LP;
3. show the practical usefulness and increased efficiency of reasoning within LP.

The organization of this part follows these three steps: Chapter 4 analyzes how Description Logic and Logic Programming are related and identifies the fragment which can be translated from one language into the other.

Chapter 5 on page 113 utilizes the identified fragment for the definition of a family of new Description Logics $\mathcal{L}_i$ and presents a semantics for these languages, which is based on a translation into Logic Programs. Moreover, the chapter shows how prototypical Description Logic reasoning problems can be reduced to answering queries on Logic Programs and studies the expressiveness and the complexity of the new languages. In particular, we show the practical usefulness of the languages by demonstrating that most parts of currently available Web ontologies can be expressed in the new languages and that the family of languages is tractable.

Chapter 6 on page 145 presents incremental maintenance of materialized ontologies (Volz et al., 2003g) as an approach to handle updates to $\mathcal{L}_i$ knowledge bases. We take the two assumptions that *(I)* updates to knowledge bases in Semantic Web application are batch and *(II)* less frequent than queries. We therefore materialize the knowledge base and handle updates to the knowledge base by incremental maintenance of the materialization.

4. The Relation between Description Logics and Logic Programming

"Let it be assumed that the states by virtue of which
the soul possesses truth by way of affirmation or denial
are five in number, i.e. art, scientific knowledge,
practical wisdom, philosophic wisdom, intuitive reason;
we do not include judgement and opinion
because in these we may be mistaken."
Aristoteles
(*Nicomachean Ethics, Book VI, Idea 3, Paragraph 1*)

This chapter explores which subsets of the OWL DL ontology language, i.e. the $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$ Description Logic (DL), can be translated into different Logic Programming (LP) environments. It formalizes and extends the analyses of (Volz et al., 2002a; Grosof et al., 2003; Volz et al., 2003b). The results of our exploration are used in chapter 5 to define a family of DL languages, for which a translation into a particular variant of Logic Programming can be guaranteed.

The chapter is organized as follows: Section 4.1 motivates the translation and discusses related work. Section 4.2 on page 82 introduces our approach for the translation from DL to LP and the identification of the LP subsets of DLs. Section 4.3 on page 83 presents syntactic normalization of DL axioms as a preprocessing technique, which simplifies the translation and maximizes the number of DL axioms that can be translated. Section 4.4 on page 87 establishes First-Order Logic (FOL) as a basic medium for a translation between DL and LP and discusses the relation of both languages with FOL itself. Section 4.5 on page 93 explores whether a lossless translation from given DL axioms to LP rules via intermediate FOL formulae is possible and precisely describes the translation and necessary transformations. Section 4.6 on page 110 summarizes the contribution of this chapter and establishes links to the subsequent chapters in this part.

4.1. Introduction

4.1.1. Motivation

A translation between Description Logic (DL) and Logic Programming (LP) establishes a correspondence between two fields of knowledge representation that are largely disparate at the time being. This correspondence will allow to transfer results, e.g. reasoning techniques, from one field into the other. This synergy can nourish the advance of both fields. For example, our technique for materializing DL knowledge bases (cf. Chapter 6 on page 145) makes such a transfer.

Besides this theoretical interest, there are two main motivations for our work:

- We need to formally identify the subset of OWL with which we can reason in logic databases, since the theoretical complexity of reasoning with OWL directly shows that a complete translation of reasoning problems is impossible.
- We want to have an a-priori guarantee on the efficiency and scalability of computing views and query answers, as well as computing the solutions to standard ABox reasoning problems. However, no scalable algorithms for ABox reasoning are yet known for expressive Description Logics, such as OWL Lite and OWL DL.

We therefore use LP as a possible formal basis for Web ontology languages and provider of efficient and scalable reasoning algorithms. Our decision is grounded on the theoretical results on the complexity of Datalog (cf. Section 2.3.5 on page 27), which show that the complexity of answering queries, viz. the main inference relevant for view management, is polynomial.

4.1.2. Related Work

Two approaches, CARIN (Levy & Rousset, 1996) and $\mathcal{AL}$-log (Donini et al., 1998), can be considered as related work, since they provide a direct extension of a particular DL with Datalog rules. However, they do not directly address our problem of translating DL to LP. They rather extend a given DL with a rule component. In order to retain the decidability of the extended language, the underlying DL is kept simple ($\mathcal{AL}$-log (Donini et al., 1998)) or restrictions are imposed on the form of rules (CARIN (Levy & Rousset, 1996)). Moreover, both approaches create intractable Description Logics. Consequently, both cannot be expressed directly in LP and require specialized reasoners, which are not publicly available.

Several other approaches do not target an extension with rules but try to reason with DL ontologies in LP environments. These approaches are based on an axiomatization of the DL primitives, i.e. they introduce special predicates, whose extension captures TBox and ABox data, and axiomatize these predicates in some way. For example, an axiomatisation of DAML+OIL, the direct precursor of OWL, was given by McGuinness and Fikes in (Fikes & McGuiness, 2001). Their axiomatization is based on the Knowledge Interchange Format (KIF) (Genesereth & Fikes, 1992).[1] Therefore, the axiomatization is not directly executable in Logic Programming systems, but rather in theorem provers. (Zou, 2001) address, however, how their axiomatization could be executed using a Prolog dialect by ignoring some axioms.[2] Besides being slightly outdated, the approach of (Fikes & McGuiness, 2001) is incomplete and most likely not sound. For example, neither the substitutivity nor all algebraic axioms of equivalence are captured.

Similarly, the two N3 systems Euler (Roo, 2002) and CWM (Berners-Lee, 2000a) try to reason with OWL via an axiomatization of RDF statements using the OWL vocabulary. Again their approach is incomplete, for example (Roo, 2002) does not correctly capture the substitutivity of equality. Additionally, no proof that their axiomatization is sound has been given yet.

(Borgida & Brachman, 1993) has focused on pushing parts of reasoning to query processing in relational databases with the intention to store the ABox in a database. This approach, although slightly related, has different objectives, i.e. storage, while taking the Closed World Assumption which is incompatible with OWL and still requiring processing outside of the database.

4.1.3. Assumptions

Axiomatization is problematic There are two main problems with axiomatization and the above mentioned approaches. Firstly, it is obviously even difficult to prove soundness, which is usually easy, as no such proof has been given yet. Secondly, even elementary constructs like subsumption, which can be understood as logical implication in LP, have to be axiomatized.

For example, we have to introduce a predicate $SubClassOf$ and axiomatize the transitivity of $SubClassOf$ by a set of rules. Therefore, the built-in optimizations provided by Logic Programming systems for elementary constructs cannot be used.

Incompleteness is unproblematic Since we want to identify a subset of the language in the first place, incompleteness is not a problem but a goal. The incom-

[1] Cf. `http://www-ksl.stanford.edu/knowledge-sharing/kif/`.

[2] Obviously, some KIF rules of the axiomatization can not be captured, e.g. Ax 105 and Ax 128.

pleteness, however, should be *precisely* characterized. This characterization is the main contribution of this chapter. Moreover, an implementation of the language within logic databases must be sound, viz. all given answers should be correct.

4.2. Approach

This section presents our general approach to the translation of a DL $\mathcal{L}_{In}$ to LP. Our approach is grounded on two fundamental ideas: Firstly, we rely on the well-known semantics and evaluation procedures for Logic Programs and, secondly, we extensively reuse available LP constructors in the translation. This should make it more easy to show soundness and should allow us to seamlessly use the optimizations built into LP systems.

4.2.1. Semantic Correspondence

Our first idea requires that we understand the correspondence between the semantics of DLs and the semantics of LP. We develop this understanding by relying on the correspondence between DL and FOL and the correspondence between FOL and LP.

4.2.2. Translation

The second idea requires us to show how a particular DL axiom can be translated into LP rules. By looking at the different complexity results for DLs and LP, we can tell beforehand that not all DL axioms can be translated. We therefore show by "affirmation or denial" whether a given constructor of a DL $\mathcal{L}_{In}$ can be translated, when situated at a certain position of an $\mathcal{L}_{In}$ axiom. It will be sufficient to distinguish two situations, namely whether a $\mathcal{L}_{In}$ constructor occurs on the left or right-hand side of inclusion axioms.

The "affirmation or denial" of the possible translation is carried out in three steps:

1. *Translate* the given $\mathcal{L}_{In}$ axiom into a logically equivalent FOL formula φ (cf. the subsequent Section 4.4);
2. *Transform* φ into skolem standard form φ' (cf. Section 4.5 on page 93);
3. *Determine* for every conjunct in φ' whether it is a Horn formula (cf. Section 4.5 on page 93).

4.2.3. Logic Programming variants

The fact that several LP variants impose restrictions on the form of Horn formulae, requires further distinctions. We consider the following important variants of Logic Programming languages as possible target logics, since they are often supported by logic databases:

- Datalog ($\mathcal{LP}_0$), which generally disallows unsafe Horn formulae, Horn formulae with empty head, the use of equality and the use of function symbols other than constants;
- Datalog(=) ($\mathcal{LP}_1$), generalizes Datalog ($\mathcal{LP}_0$) by allowing the use of equality;
- Datalog(=,IC) ($\mathcal{LP}_2$), generalizes Datalog(=) ($\mathcal{LP}_1$) by allowing Horn formulae with empty head (so-called *integrity constraints*);
- Prolog(=,IC) ($\mathcal{LP}_3$), generalizes Datalog(=,IC) ($\mathcal{LP}_2$) by allowing function symbols and unsafe Horn formulae.

The individual logics are upwards compatible and of increasing expressivity. Every $\mathcal{LP}_i$ program is also an $\mathcal{LP}_{(i+1)}$ program and, more importantly, is interpreted in exactly the same way in all $\mathcal{LP}_{(i+1)}$.

4.3. Preprocessing Techniques

This section describes three techniques that reduce the total number of constructor combinations that we have to consider in the following and turn out to maximize the number of axioms that can be translated into $\mathcal{LP}$. Two techniques carry out a *normalization* and *simplification* of class descriptions and axioms. The third technique is structural transformation, which is not necessarily required, but allows us to concentrate on individual DL constructors and not arbitrary complex class descriptions.

The first two techniques are motivated by the fact that expressive Description Logics such as $\mathcal{SHIN}\mathcal{O}$ often expose a certain redundancy in the set of its constructors. For example, logics with full negation often provide pairs of operators of which either one can be expressed as the other by using negation. We therefore will consider such tautologies (cf. Table 4.1, where $\rightrightarrows$ is used to denote the equivalence and the direction of application) to avoid redundant work in our translation.

It is important to note that it is useful to have this redundancy from an epistemological perspective, since it is much more natural for a human knowledge engineer to

$\neg\forall R.C$	$\equiv$	$\exists R.\neg C$	$\neg\exists R.C$	$\equiv$	$\forall R.\neg C$	$\leq n\, R$	$\equiv$	$\neg\geq (n+1)\, R$
$\neg(\sqcap_i C_i)$	$\equiv$	$\sqcup_i(\neg C_i)$	$\neg(\sqcup_i C_i)$	$\equiv$	$\sqcap_i(\neg C_i)$	$\neg\top$	$\equiv$	$\bot$
$\top \sqcap C$	$\equiv$	C	$\bot \sqcap C$	$\equiv$	$\bot$	$\forall R.\top$	$\equiv$	$\top$
$\top \sqcup C$	$\equiv$	$\top$	$\bot \sqcup C$	$\equiv$	C	$\exists R.\bot$	$\equiv$	$\bot$
$\leq 0\, R$	$\equiv$	$\forall R.\bot$	$\geq 1\, R$	$\equiv$	$\exists R.\top$	$\geq 0\, R$	$\equiv$	$\top$

Table 4.1.: Equivalences for Normalization/Simplification of Class Descriptions

state a disjunction of classes than the negation of a conjunction of negated classes. Similarly, humans typically do not engineer knowledge bases with the focus that subsequent reasoning is faster.

Modern DL systems therefore normalize and simplify axioms using preprocessing techniques to speed up reasoning by avoiding the encoding of (parts of) axioms that are logically unnecessary. For example, one can detect certain contradictions syntactically. We adapt these techniques for the two aforementioned purposes.

Firstly, we transform class descriptions into a form that can be translated into $\mathcal{LP}$ more easily. Secondly, we remove redundancies to reduce our effort of determining the intersection.

4.3.1. Normalization/Simplification of class descriptions

Our normalization and simplification of class descriptions is different from the normalization and simplification carried out in modern DL reasoners. In DL reasoners, normalization transforms expressions into *negation normal form* by pushing negation in an expression towards atomic class names while reducing the language features. For example, in $\mathcal{ALC}$ all occurrences of $\exists$ and $\sqcup$ would be completely removed and always be replaced by $\neg\forall$ and $\neg\sqcap$.

In our case, we do not completely remove the constructs, but incorporate basic equivalences such as the DeMorgan laws and other equivalences that we have presented in Table 4.1. In fact, our particular choice of direction in the application of these equivalences is motivated by the results of our exploration in Section 4.5. In particular, we try to eliminate negation as far as possible by pushing negation in an expression towards atomic class names. Unlike the normalization used to construct the negation normal form, we do, however, not introduce negation by replacing constructors.

Example 4.3.1 *As we will see later, the following axiom cannot be translated into $\mathcal{LP}$, if*

$$
\begin{array}{rcl}
\text{Norm}(A) & = & \textit{A for atomic class } A \\
\text{Norm}(\sqcap_i C_i) & = & \text{Simp}(\sqcap_i \{\text{Norm}(C_1) \ldots \{\text{Norm}(C_n)\}\}) \\
\text{Norm}(\sqcup_i C_i) & = & \text{Simp}(\sqcup_i \{\text{Norm}(C_1) \ldots \{\text{Norm}(C_n)\}\}) \\
\text{Norm}(\forall R.C) & = & \text{Simp}(\forall R.\text{Norm}(C)) \\
\text{Norm}(\exists R.C) & = & \text{Simp}(\exists R.\text{Norm}(C)) \\
\text{Norm}(\{o_1, \ldots, o_n\}) & = & \{o_1, \ldots, o_n\} \\
\text{Norm}(\geq n\, R) & = & \text{Simp}(\geq n\, R) \\
\text{Norm}(\leq n\, R) & = & \text{Simp}(\leq n\, R) \\
\text{Norm}(\neg C) & = & \text{Simp}(\neg \text{Norm}(C))^{(A)} \\
\text{Norm}(\neg \sqcap_i C_i) & = & \text{Simp}(\sqcup_i \{\text{Norm}(\neg C_1) \ldots \{\text{Norm}(\neg C_n)\}\}) \\
\text{Norm}(\neg \sqcup_i C_i) & = & \text{Simp}(\sqcap_i \{\text{Norm}(\neg C_1) \ldots \{\text{Norm}(\neg C_n)\}\}) \\
\text{Norm}(\neg \forall R.C) & = & \text{Simp}(\exists R.\text{Norm}(\neg C)) \\
\text{Norm}(\neg \exists R.C) & = & \text{Simp}(\forall R.\text{Norm}(\neg C)) \\
\text{Norm}(\neg \geq n\, R) & = & \text{Simp}(\leq n-1\, R) \\
\text{Norm}(\neg \leq n\, R) & = & \text{Simp}(\geq (n+1)\, R)
\end{array}
$$

$$
\begin{array}{rcl}
\text{Simp}(A) & = & \textit{A for atomic class } A \\
\text{Simp}(\neg C) & = & \bot, \text{ if } C = \top \\
 & & \top, \text{ if } C = \bot \\
 & & \text{Simp}(D), \text{ if } C = \neg D \\
 & & \neg C, \text{ otherwise} \\
\text{Simp}(\sqcap \mathbf{S}) & = & \bot, \text{ if } \bot \in \mathbf{S} \\
 & & \bot, \text{ if } \{C, \neg C\} \subseteq \mathbf{S} \\
 & & \top, \text{ if } \mathbf{S} = \emptyset \\
 & & \text{Simp}(\mathbf{S} \setminus \{\top\}), \text{ if } \top \in \mathbf{S} \\
 & & \sqcap \mathbf{S}, \textit{otherwise} \\
\text{Simp}(\sqcup \mathbf{S}) & = & \top, \text{ if } \top \in \mathbf{S} \\
 & & \top, \text{ if } \{C, \neg C\} \subseteq \mathbf{S} \\
 & & \bot, \text{ if } \mathbf{S} = \emptyset \\
 & & \text{Simp}(\mathbf{S} \setminus \{\bot\}), \text{ if } \bot \in \mathbf{S} \\
 & & \sqcup \mathbf{S}, \text{ otherwise} \\
\text{Simp}(\forall R.C) & = & \top, \text{ if } C = \top \\
 & & \forall R.C, \text{ otherwise} \\
\text{Simp}(\exists R.C) & = & \bot, \text{ if } C = \bot \\
 & & \exists R.C \\
\text{Simp}(\geq n\, R) & = & \top, \text{ if } n = 0 \\
 & & \exists R.\top, \text{ if } n = 1 \\
 & & \geq n\, R, \text{ otherwise} \\
\text{Simp}(\leq n\, R) & = & \bot, \text{ if } n = -1^{(B)} \\
 & & \forall R.\bot, \text{ if } n = 0 \\
 & & \leq n\, R, \text{ otherwise}
\end{array}
$$

(A) Defeasible rule, which is only used if the other negation rules cannot be applied.

(B) Required for normalization of negation.

Table 4.2.: Normalization and Simplification Rules

we only consider the structure of the axiom:

$$C \sqsubseteq \neg{\geq}\,0\,P \sqcup (\geq 1\,R \sqcap \neg(\exists R.(\neg A \sqcup \neg B)))$$

Simplification and normalization, however, lead to the following logically equivalent expression, for which we will learn that it can be translated into $\mathcal{LP}$:

$$C \sqsubseteq \exists R\top \sqcap \forall R.A \sqcap B$$

The simplification is achieved by the recursive application of the normalization and simplification functions described in Table 4.2 on page 85:

$\neg{\geq}\,0\,P \sqcup (\geq 1\,R \sqcap \neg(\exists R.(\neg A \sqcup \neg B)))$
$\mathit{Simp}(\sqcup\{\mathit{Norm}(\neg{\geq}\,0\,P), \mathit{Norm}((\geq 1\,R \sqcap \neg(\exists R.(\neg A \sqcup \neg B))))\})$
$\mathit{Simp}(\sqcup\{\mathit{Simp}(\leq -1\,P)\}, \mathit{Simp}(\sqcap\{\mathit{Norm}(\geq 1\,R), \mathit{Norm}(\neg(\exists R.(\neg A \sqcup \neg B)))\}))$
$\mathit{Simp}(\sqcup\{\mathit{Simp}(\bot), \mathit{Simp}(\sqcap\{\mathit{Simp}(\geq 1\,R), \mathit{Simp}(\forall R.\mathit{Norm}(\neg(\neg A \sqcup \neg B)))\})\})$
$\mathit{Simp}(\sqcup\{\bot, \mathit{Simp}(\sqcap\{\exists R\top, \mathit{Simp}(\forall R.\mathit{Simp}(\sqcap\{\mathit{Norm}(A), \mathit{Norm}(B)\}))\})\})$
$\mathit{Simp}(\sqcup\{\bot, \mathit{Simp}(\sqcap\{\exists R\top, \mathit{Simp}(\forall R.\mathit{Simp}(\sqcap\{A, B\}))\})\})$
$\mathit{Simp}(\sqcup\{\bot, \mathit{Simp}(\sqcap\{\exists R\top, \mathit{Simp}(\forall R. \sqcap \{A, B\})\})\})$
$\mathit{Simp}(\sqcup\{\bot, \mathit{Simp}(\sqcap\{\exists R\top, \forall R. \sqcap \{A, B\}\})\})$
$\mathit{Simp}(\sqcup\{\bot, \sqcap\{\exists R\top, \forall R. \sqcap \{A, B\}\}\})$
$\sqcap\{\exists R\top, \forall R. \sqcap \{A, B\}\}$

An important refinement that we also include here, is to treat n-ary class constructors as sets, which eases implementation (cf. Section 7.4 on page 180). The simplification rules also try to detect basic inconsistencies within these sets. For example, if $\bot$ is included in a conjunction, then the whole set evaluates to $\bot$. Also, if C and $\neg C$ are contained in the set, we can deduce $\bot$ automatically. Obviously, the opposite simplifications can be made for disjunctions.

Empirically, simplification and normalization showed to be very effective for satisfiability problems in large or mechanically generated TBoxes. For example, some classes of problem in the Tableaux 98 benchmark comparison of modal logic theorem provers could completely be solved using structural simplification and normalization (Balsiger & Heuerding, 1998).

In our case, simplification and normalization turn out to be very useful in the Semantic Web setting, where we are likely to encounter OWL ontologies that have been designed by others who might not have given consideration to whether the designed ontologies can be directly used in logic databases.

4.3.2. Normalization of axioms

In some cases we are able to remove negation from axioms. Two pre-processing optimizations deal with these situations and operate on an axiom level.

Firstly, we can use the following equivalence

$$C \sqsubseteq D \sqcup \neg N_1 \sqcup \ldots \sqcup \neg N_n \Leftrightarrow C \sqcap N_1 \sqcap \ldots \sqcap N_n \sqsubseteq D$$

to translate disjunctions on the right-hand side of an axiom, if the disjunction contains at most one positive disjunct.

Secondly, we can use the following equivalence

$$\neg C \sqsubseteq \neg D \Leftrightarrow D \sqsubseteq C$$

to translate inclusion and equivalence axioms where negation occurs on the outer side of the class descriptions on both sides of the axiom.

4.3.3. Structural Transformation

For complex class descriptions we apply structural transformation (Plaisted & Greenbaum, 1986), also known as renaming, i.e. assign a new name for each instantiation of a DL constructor in a (complex) class description and axiomatize this name via a separate inclusion axiom.

Example 4.3.2 (Structural Transformation) *Consider the axiom $\exists P.A \sqsubseteq \forall Q.B$, where A, B (P, Q) are atomic class (property) names. Structural transformation produces three axioms using the previously unused class identifiers N_1 and N_2:*

$$\{\exists P.A \sqsubseteq N_1, N_2 \sqsubseteq \forall Q.B, N_1 \sqsubseteq N_2\}$$

We can thereby restrict our attention to the translation of individual DL constructors. Obviously, since the number of constructors used in a complex DL class description C is linear in the size of C, we only introduce a linear number of new DL axioms.

4.4. From DL to FOL to LP

We can now turn to the core of our approach and start with the analysis of the semantic correspondence between DL and LP. As mentioned before, we use FOL as a mediator between the two logics. Our analysis of the correspondence between DL and FOL in the following subsection, will also address step 1 of our approach (cf. Section 4.2 on page 82), viz. the *translation* of a given Description Logic $\mathcal{L}_{In}$ into FOL. To this extent, we equate $\mathcal{L}_{In}$ with the Description Logic underlying OWL DL and ignore the aspects of data typing. Therefore, we are actually concerned with

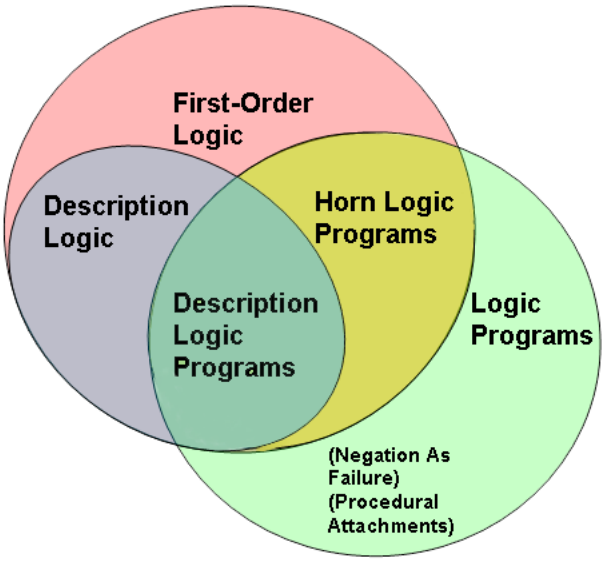


Figure 4.1.: Expressive Overlap of DL with LP.

$\mathcal{L}_{In}$=$\mathcal{SHIN}\mathcal{O}$. Section 4.4.2 on page 92 will then consider how FOL and LP are related with each other.

Figure 4.1 on page 88 illustrates the relationship between First Order Logic (FOL), Horn Logic, Description Logics and Logic Programs. Both Description Logic and function-free Horn Logic are strict (decidable) subsets of FOL. The decidability, however, has been achieved by making different syntactic restrictions:

- Function-free Horn Logic basically only allows Horn formulae (cf. Definition 2.2.26 on page 19) and disallows function-symbols in term construction.
- Description Logics limit the way how variables and quantifiers can be used. In particular, quantifiers must be *relativized* via atomic formulae (as in the guarded fragment of FOL (Grädel, 1999)), i.e. the quantified variable must occur in a property predicate along with the free variable.

Our approach to translation can therefore also be formulated as determining the correspondence between these two restrictions.

4.4.1. From DL to FOL

Description Logics can be understood as a notational variant of a fragment of FOL. To this extent, classes (properties) can be understood as unary (binary) predicates,

since $\mathfrak{I}$ interprets them as unary and binary relations over $\Delta^{\mathfrak{I}}$.

We can therefore translate each class description C into a FOL formula $\phi_C(x)$ with one free variable such that for every interpretation $\mathfrak{I}$, elements of $\Delta^{\mathfrak{I}}$ satisfy $\phi_C(x)$ iff they satisfy $C^{\mathfrak{I}}$. To be more precise, we conceptually follow (Borgida, 1996)[3] and define:

Definition 4.4.1 (Equivalence of DL Class to FOL formula) *A class C and its translation $\phi_C(x)$ to a FOL formula are equivalent if we have for all interpretations $\mathfrak{I} = (\Delta^{\mathfrak{I}}, \cdot^{\mathfrak{I}})$ and all $a \in \Delta^{\mathfrak{I}}$:*

$$a \in C^{\mathfrak{I}} \text{ if } \mathfrak{I} \models \phi_C(a).$$

4.4.1.1. Translation of Class Descriptions

(Borgida, 1996) inductively defines a translation[4] that obeys the above definition and translates all DL-classes and DL-properties into equivalent FOL formulae.

We simplify the translation of (Borgida, 1996), such that multiple variables are used[5]:

- $\phi_x(A)$ produces an unary predicate $A(x)$ whose free variable is x for its argument class A;
- $\phi_{(x,y)}(R)$ produces, for its argument property R, a binary predicate $R(x, y)$ whose free variables are x and y;
- The mapping function $\phi_{(x,y,z)}(R^+ \sqsubseteq R)$ partially[6] captures transitive properties and produces, for its argument axiom, a FOL sentence

$$\forall x, y, z.(R(x, y) \wedge R(y, z)) \rightarrow R(x, z)$$

The inductive definition of the translation functions for our particular $\mathcal{L}_{In}$, viz. $\mathcal{SHIN}\mathcal{O}$, is given in Table 4.3 (for classes) and Table 4.4 (for properties). The trans-

[3] We weaken the equivalence of (Borgida, 1996) to satisfiability only, which is sufficient for DL reasoning problems and OWL entailment.

[4] It parallels the translation of propositional modal logic into FOL given by (van Benthem, 1984).

[5] (Borgida, 1996) constructs the translation using two variables only, which is used for showing that the resulting FOL formulae are expressible in $\mathcal{L}^2$, the subset of first-order formulae with no function symbols and maximum two variables, which is known to be decidable (Graedel et al., 1997).

[6] As a consequence of the compactness theorem of first-order logic, we cannot express in FOL that P^+ is indeed the smallest transitive relation including P.

DL	FOL
$\phi_x(A)$	$A(x)$
$\phi_x(C \sqcap D)$	$\phi_x(C) \wedge \phi_x(D)$
$\phi_x(C \sqcup D)$	$\phi_x(C) \vee \phi_x(D)$
$\phi_x(\neg C)$	$\neg\phi_x(C)$
$\phi_x(\exists R.C)$	$\exists y.\phi_{(x,y)}(R) \wedge \phi_y(C)$
$\phi_x(\forall R.C)$	$\forall y.\phi_{(x,y)}(R) \rightarrow \phi_y(C)$
$\phi_x(\{i_1 \ldots i_n\})$	$x = i_1 \vee \ldots \vee x = i_n$
$\phi_x(\top)$	$v = v$
$\phi_x(\bot)$	$\neg(x = x)$
$\phi_x(\geq n\, R)$	$\exists y_1 \ldots y_n. \bigwedge_{i \neq j} \neg(y_i = y_j) \bigwedge_i R(x, y_i)$
$\phi_x(\leq n\, R)$	$\forall y_1 \ldots y_{n+1}. \bigwedge_i R(x, y_i) \rightarrow \bigvee_{i \neq j}(y_i = y_j)$

Table 4.3.: FOL Translation of $\mathcal{SHINO}$ Class Descriptions

lation of classes is recursive and introduces new (previously unused) variables in the translation of restrictions[7].

Example 4.4.1 *The DL class description $C \sqcap \forall R.D$ would be translated by ϕ_x into the following FOL formula:*

$$\phi_x(C \sqcap \forall R.D) \Leftrightarrow \phi_x(C) \wedge \phi_x(\forall R.D) \Leftrightarrow C(x) \wedge (\forall y.\phi_{(x,y)}(R) \rightarrow \phi_y(D)) \Leftrightarrow C(x) \wedge (R(x,y) \rightarrow D(y))$$

DL	FOL
$\phi_{(x,y)}(R)$	$\forall x, y.R(x, y)$
$\phi_{(x,y,z)}(R^+ \sqsubseteq R)$	$\forall x, y, z.(R(x, y) \wedge R(y, z)) \rightarrow R(x, z)$
$\phi_{(x,y)}(R^-)$	$\phi_{(y,x)}(R)$

Table 4.4.: FOL Translation of $\mathcal{SHINO}$ Property Axioms

Theorem 4.4.1 (Class Satisfiability) *A class C is satisfiable if $\phi_x(C)$ is satisfiable.*

Proof Sketch: Let the universe of discourse $\Delta^{\mathfrak{I}}$ of the DL be the domain of $\mathfrak{I}$. As usual in FOL, the variable valuation function $\mathcal{V} : V \rightarrow \Delta^{\mathfrak{I}}$ assigns an element

[7] An alternative that avoids the large number of inequalities that arise from the translation presented in Table 4.3 would translate number restrictions into counting quantifiers, viz. an extension of FOL with such quantifiers. Since such quantifiers are not available in LP, this variant is not regarded here.

$d \in \Delta^{\mathfrak{I}}$ to every variable $v \in V$. We choose the denotation of predicates to assign $[C(x)]^{\mathcal{V}}_I = true$ iff $\mathcal{V}(x) \in C^{\mathfrak{I}}$, viz. we assign the DL interpretation $P^{\mathfrak{I}}$ as interpretation of each predicate symbol P. The proof is then completed by structural induction over the formula (cf. (Borgida, 1996)).

So far we were only concerned with meaning preserving translations of classes and properties into logical formulae, thereby reducing class satisfiability to satisfiability of formulae in FOL.

4.4.1.2. Translation of Knowledge Bases

Description Logics also allow us to state a TBox $\mathcal{T}$. We can then determine whether a class C is satisfiable with respect to the semantic conditions stated by the axioms in $\mathcal{T}$. The equivalence of this procedure in FOL is very simple.

Firstly, all axioms in $\mathcal{T}$ are translated into one big universally quantified conjunction where each conjunct corresponds to an individual axiom. We expand $C \equiv D$ into two axioms $\{C \sqsubseteq D, D \sqsubseteq C\}$. This allows us to deal with general inclusion axioms only. We apply the same strategy to axioms which state the equivalence of properties, viz. $P \equiv Q$ will be expanded into two axioms $\{P \sqsubseteq Q, Q \sqsubseteq P\}$.

Given a TBox $\mathcal{T} = \{C_i \sqsubseteq D_i | 1 \leq i \leq n\} \cup \{P_j \sqsubseteq Q_j | 1 \leq j \leq m\}$ where C_i, D_i are classes and P_j, Q_j are properties, we define:

$$\phi(\mathcal{T}) = \forall x.(\bigwedge_{i=1}^{n}(\phi_x(C_i) \rightarrow \phi_x(D_i))) \wedge \forall x, y.(\bigwedge_{j=1}^{m}(\phi_{x,y}(P_j) \rightarrow \phi_{x,y}(Q_j))) \tag{4.1}$$

(Borgida, 1996) shows the following theorem:

Theorem 4.4.2 (Satisfiability w.r.t. a TBox) *A class C is satisfiable with respect to $\mathcal{T}$ iff the formula $\phi(\mathcal{T}) \wedge \phi_x(C)$ is satisfiable.*

All other reasoning problems for Description Logics (cf. Section 2.4.3 on page 38) can be translated to a single FOL formula in a similar fashion.

DL	FOL
$C(i)$	$\phi_i(C)$
$R(a,b)$	$R(a,b)$
$i = j$	$i = j$
$i \neq j$	$\neg(i = j)$

Table 4.5.: FOL Translation of ABox Assertions

Assertional axioms Table 4.5 shows the translation of individual assertions. Individual assertions are translated by instantiating ϕ using a constant i that represents the individual i in the DL ABox. We can check the satisfiability of an ABox $\mathcal{A}$, which may contain property assertions, class assertions and assertions stating the (in)equivalence of individuals, with respect to a TBox $\mathcal{T}$ by translating the set of assertions into a single FOL conjunction. Given an ABox $\mathcal{A}$ with

$$\mathcal{A} = \{R_k(a_i, a_j)|(i,j,k) \in I)\} \cup \{C_j(a_k)|(i,j) \in J\} \cup \{a_i = a_j|(i,j) \in K\} \cup \{\neg(a_i = a_j)|(i,j) \in L\}$$

where I, J, K, L are simple index sets and all a_i are individuals in the ABox, we can define

$$\phi(\mathcal{A}) = \bigwedge_{(i,j,k \in I)} R_k(a_i, a_j) \wedge \bigwedge_{(i,j) \in J} \phi_{a_i}(C_j) \wedge \bigwedge_{(i,j) \in K} a_i = a_j \wedge \bigwedge_{(i,j) \in L} \neg(a_i = a_j) \quad (4.2)$$

by translating individuals into constants in FOL. The consistency of $\mathcal{A}$ with respect to $\mathcal{T}$ can then be determined by checking whether the formula $\phi(\mathcal{A}) \wedge \phi(\mathcal{T})$ is satisfiable. We can determine whether a is an instance of C by checking whether $\phi(\mathcal{A}) \wedge \phi(\mathcal{T}) \wedge \neg\phi_a(C)$ is unsatisfiable. All other reasoning problems for ABoxes (cf. Section 2.4.3 on page 38) can be translated similarly, viz. checking (un)satisfiability of a single FOL formula suffices to solve all DL reasoning problems.

4.4.2. From FOL to LP

The relationship of First-Order Logic to Logic Programs is less clear. When programs are definite and do not contain negation-as-failure, Logic Programs syntactically correspond to Horn formulae. However, the evaluation procedures of logic databases typically only calculate the minimal Herbrand model of the set of Horn formulae (cf. Section 2.3.2 on page 21), i.e. compute the least fixpoint, and entail only facts, i.e. unit clauses.

The least fixpoint computation itself is not expressible in FOL and the limitation to entailment of unit clauses means that logic databases do not make conclusions about the program itself and do not entail new rules. This is, however, possible for Horn formulae. For example, one can entail from the formulae $sunny(Tuesday) \wedge windy(Tuesday) \rightarrow kiting(Tuesday)$ and $sunny(Tuesday)$ that $windy(Tuesday) \rightarrow kiting(Tuesday)$ holds. The latter derivation is a non-unit clause and can therefore not be produced by logic databases.

For these reasons, definite Logic Programs can be understood as an *entailment-weakened* form of Horn formulae. This weakening of entailment is not problematic, since we do not require the entailment of rules (such a reasoning problem is usually not considered for Description Logics).

Moreover, several features of Logic Programs (cf. Figure 4.1 on page 88), which are frequently used in practical rule-based applications, are inexpressible in FOL (and consequently also outside of the Horn-fragment of FOL). One example is negation-as-failure, a basic kind of logical non-monotonicity. Another example is procedural attachments, e.g. the association of action-performing procedural invocations with the drawing of conclusions about particular predicates.

4.5. Finding the Intersection

This section is concerned with the steps 2 and 3 of our approach, viz. *transforming* the FOL formula obtained from the translation in the previous section into skolem standard form and *determining* for every conjunct whether it is a Horn formula. We thereby decide, whether a language construct in $\mathcal{L}_{In} = \mathcal{SHINO}$ can be translated into a $\mathcal{LP}$ variant. The impatient reader may refer to Table 4.7 on page 111 which summarizes our findings.

The subsequent sections will explore all (normalized) $\mathcal{SHINO}$ constructors and axioms in detail. We will state several simple corollaries and fully give (trivial) proofs since we will refer to the conclusions and transformations established here in subsequent sections.

We will use the following theorem to show that a particular formula *can not* be translated into Logic Programs:

Theorem 4.5.1 *A non-tautological Gentzen formula with more than one positive literal cannot be reduced to a single set of Horn formulae.*

Proof: A non-tautological Gentzen formula with more than one positive literal has more than one minimal model, all of which characterize its logical consequences. Since a set of Horn formula only has one minimal model, the set cannot possibly capture all logical consequences of a non-tautological Gentzen formula with more than one positive literal.

□

Example 4.5.1 (Multiple Minimal Models for Gentzen formulae) *The Gentzen formula $P(a) \vee P(b)$, where a and b are constants, has $\{\{P(a)\}, \{P(b)\}\}$ as a non-empty set of models. The intersection of the two models is empty, i.e. a valid Herbrand interpretation, but not a model. (From (van Emden & Kowalski, 1976))*

Remark 4.5.1 *The following proofs will provide a transformation to a conjunction of disjunctions. If every disjunction is a Horn formula, the translation into a $\mathcal{LP}$ is straightforward, i.e. the conjunction corresponds to a set of LP rules. Since this transformation is purely syntactic, we will omit it in the proofs.*

4.5.1. Assertions

Description Logic ABox assertions are (unit) clauses in their FOL notation, viz. they are Horn formulae and it is directly clear how they are represented as LP facts. Note that almost all ground constants in the logic program are introduced via such ABox assertions.

4.5.1.1. Anonymous Individuals

We assume that all occurrences of anonymous individuals have been replaced by skolem constants via the reduction described in Section 3.4.3.2 on page 66. Note, however, that an implementation must be able to distinguish between "real" individuals and skolem constants.

4.5.1.2. Individual assertions

All ABox individual assertions on atomic classes $A(a)$ are translated to unit clauses $A(a)$. Similarly, all ABox property fillers are translated to unit clauses $P(a, b)$. Assertions on complex classes C are handled by introducing a *new* atomic class name I_i. The ABox assertion $C(a)$ is then replaced by a new ABox assertion $I_i(a)$ and a new axiom $I_i \sqsubseteq C$ is introduced into the TBox. In an implementation, I_i is a system-internal class (in style of the names generated by structural transformation) and may not necessarily be shown to the user.

Example 4.5.2 *The individual assertion*

$$\exists \text{HASCHILD.MAN}(\text{JOHANN-AMBROSIUS})$$

from Table 2.5 on page 35 is represented as follows

$$I_1(\text{JOHANN-AMBROSIUS})$$
$$I_1 \sqsubseteq \exists \text{HASCHILD.MAN}$$

Individual equality Individual equality axioms can only be asserted in a $\mathcal{LP}$ that allows an equality predicate, viz. $\mathcal{LP}_1$ or above.

Individual inequality Individual inequality axioms can only be asserted in a $\mathcal{LP}$ that allows an equality predicate and integrity constraints, viz. $\mathcal{LP}_2$ or above. Every inequality assertion $\neg(a = b)$ is essentially an integrity constraint.

4.5.2. Atomic Classes and Properties

In the following, we will restrict our attention to atomic classes, viz. the symbols A_i and B_i do not refer to class descriptions but only class names. In $\mathcal{L}_{In}$, i.e. $\mathcal{SHIN}\mathcal{O}$, all properties P_i and Q_i are atomic by default.

Corollary 4.5.1 *Subsumption and equivalence of atomic classes and properties can be represented in $\mathcal{LP}_0$.*

Proof: Immediate consequence of the translation from DL to FOL (cf. Equation 4.1 on page 91):

- Class Inclusion Axioms
  $\phi(A \sqsubseteq B) = \forall x.(A(x) \rightarrow B(x))$
- Class Equivalence Axioms
  $\phi(A \equiv B) = \forall x.((A(x) \rightarrow B(x)) \wedge (B(x) \rightarrow A(x)))$
- Property Inclusion Axioms
  $\phi(P \sqsubseteq Q) = \forall x, y.(P(x, y) \rightarrow Q(x, y))$
- Property Equivalence Axioms
  $\phi(P \equiv Q) = \forall x, y.((P(x, y) \rightarrow Q(x, y)) \wedge (Q(x, y) \rightarrow P(x, y)))$

□

4.5.3. Universal class $\top$ and Empty Class $\bot$

According to Table 4.3 on page 90, $\top$ is translated into FOL by expressing the tautology $x = x$. Similarly $\bot$ is translated to a formula that can never be satisfied, viz. $\neg(x = x)$.

We can immediately see that the translation of $\top$ and $\bot$ at least needs equality.

4.5.3.1. Universal class $\top$

Corollary 4.5.2 (Equivalence $\top$) *An occurrence of $\top$ in class equivalence axioms can be represented in $\mathcal{LP}_1$.*

Proof: Immediate consequence of the translation from DL to FOL and the fact that the resulting formulae make use of equality:

- $\top$ on the left-hand side of DL inclusion axioms
  $\phi(\top \sqsubseteq C) = \forall x.((x = x) \rightarrow C(x))$
- $\top$ on the right-hand side of DL inclusion axioms
  $\phi(C \sqsubseteq \top) = \forall x.(C(x) \rightarrow (x = x))$
- $\top$ in class equivalence axioms
  $\phi(\top \equiv C) = \phi(\top \sqsubseteq C) \wedge \phi(C \sqsubseteq \top) = \forall x.(((x = x) \rightarrow C(x)) \wedge (C(x) \rightarrow (x = x)))$

□

4.5.3.2. Empty class $\bot$

Corollary 4.5.3 (right-hand side $\bot$) *An occurrence of $\bot$ on the right-hand side of DL inclusion axioms can be represented in $\mathcal{LP}_2$.*

Proof: The following transformation shows that the corresponding formula is a Horn formula with empty head, i.e. an integrity constraint:
$\phi(C \sqsubseteq \bot) = \forall x.(C(x) \rightarrow \neg(x = x)) \Leftrightarrow \forall x.(\neg C(x) \vee \neg(x = x))$

□

Corollary 4.5.4 (left-hand side $\bot$) *An occurrence of $\bot$ on the left-hand side of DL inclusion axioms and class equivalence axioms can **not** be represented in $\mathcal{LP}$.*

Proof:

- left-hand side of DL inclusion axioms
  $\phi(\bot \sqsubseteq C) = \forall x.(\neg(x = x) \rightarrow C(x)) \Leftrightarrow \forall x.(C(x) \vee (x = x))$ is a Gentzen formula. In consequence of Theorem 4.5.1 on page 93, the formula can not be translated into a set of Horn formulae.

- Class equivalence axioms follow directly from the translation of equivalence into pairs of inclusion axioms.

□

Example 4.5.3 (Basic Inconsistency) *A consequence of Corollary 4.5.3 is that we can make the whole knowledge base unsatisfiable by a single axiom:*

$$\top \sqsubseteq \bot$$

This would be translated to the following integrity constraint

$$\texttt{:-}(x = x).$$

Hence, it is impossible to create any facts, since we can deduce $x = x$ for every element in the Herbrand base ($x \in B_H$) of the $\mathcal{LP}$.

Remark 4.5.2 *Several alternative ways to encode $\top$ and $\bot$ exist*

- ***true*/*false***: *One possibility is to translate $\top$ to **true** and $\bot$ to **false**. However, a left-hand side occurrence of $\top$ would be an unsafe rule, whereas a right-hand side occurrence of top would lead to an elimination of the rule, viz. the symbol for the class could possibly disappear from the logic program. It would be impossible to detect within the logic program, whether the class has ever been declared or not. For example, the class could never be used in a user query. The alternative translation into **true*/*false** can, however, be used in implementations for an a-posteriori optimization of the logic program.*

- TOP(x)/BOTTOM(x): *One could axiomatize these two predicates, by stating that all elements of the Herbrand universe of the logic program are members of* TOP(x) *and stating the integrity constraint* :- BOTTOM(x). *However, it would be more cumbersome to prove the correctness, which immediately follows in our approach from the FOL translation and tautological transformation into a conjunction of Horn rules.*

4.5.4. Boolean Class Descriptions

A Description Logic class description can be formed by the combination of various constructors. Without loss of generality, we limit the following classes C_i, D_i to atomic class names.

4.5.4.1. Conjunction

Corollary 4.5.5 (Equivalence $\sqcap$) *A conjunction $\sqcap_i C_i$ with $C_i \neq \{\top, \bot\}$ in class equivalence axioms can be represented in $\mathcal{LP}_0$.*

Proof:

Immediate consequence of the translation of equivalence into a conjunction of two inverse inclusion axioms and the following transformations:

- $\sqcap$ on the left-hand side of DL inclusion axioms
  $\phi(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D) = \forall x.(C_1(x) \wedge \ldots C_n(x) \rightarrow D(x))$
- $\sqcap$ on the right-hand side of DL inclusion axioms
  $\phi(D \sqsubseteq C_1 \sqcap \ldots \sqcap C_n) \quad = \forall x.(\bigwedge_{i\in[1,n]}(D(x) \rightarrow C_i(x)))$
- Class equivalence axioms
  $\phi(C_1 \sqcap \ldots \sqcap C_n \equiv D) = \phi(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D) \wedge \phi(D \sqsubseteq C_1 \sqcap \ldots \sqcap C_n)$
  $\Leftrightarrow \forall x.(C_1(x) \wedge \ldots C_n(x) \rightarrow D(x))) \wedge (\bigwedge_{i\in[1,n]}(D(x) \rightarrow C_i(x))$

□

Example 4.5.4 *The class equivalence axiom*

$$\text{MALECOMPOSER} \equiv \text{COMPOSER} \sqcap \text{MALE}$$

from Table 2.5 on page 35 is represented as follows in LP:

$$\begin{aligned} \text{COMPOSER}(x) &\text{ :- } \text{MALECOMPOSER}(x). \\ \text{MALE}(x) &\text{ :- } \text{MALECOMPOSER}(x). \\ \text{MALECOMPOSER}(x) &\text{ :- } \text{COMPOSER}(x), \text{MALE}(x). \end{aligned} \tag{4.3}$$

Remark 4.5.3 ($\top$ and $\bot$ in conjunctions) *In consequence of the tautology $C \sqcap \top \equiv C$, $\mathcal{LP}_1$ is not needed to represent the conjunction, if one conjunct is $\top$. On the other hand, since $C \sqcap \bot \equiv \bot$ we need $\mathcal{LP}_2$, i.e. equality and the ability to express integrity constraints, to represent a conjunction on the right-hand side of an inclusion axiom if one conjunct is $\bot$; we cannot express conjunction on the left-hand side of the inclusion axiom nor in equivalence axioms, if a conjunct is $\bot$.*

4.5.4.2. Disjunction

Corollary 4.5.6 (left-hand side $\sqcup$) *A disjunction $\sqcup_i C_i$ with $C_i \neq \{\top, \bot\}$ on the left-hand side of DL inclusion axioms can be represented in $\mathcal{LP}_0$.*

Proof: We can apply the following transformation

$$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq D \Leftrightarrow \{C_i \sqsubseteq D\}.$$
$$\phi(\{C_i \sqsubseteq D | i \in [1,n]\}) = \bigwedge_{i=1}^{n} \phi(C_i \sqsubseteq D).$$

The conjunction can be represented in $\mathcal{LP}_0$ since every conjunct can be represented in $\mathcal{LP}_0$ by Corollary 4.5.1 on page 95.

□

Corollary 4.5.7 (right-hand side $\sqcup$) *A disjunction $\sqcup_i C_i$ with more than one disjunct $C_i \neq \{\top, \bot\}$ on the right-hand side of a DL inclusion axiom can **not** be represented in $\mathcal{LP}$.*

Proof: The following transformation shows that the corresponding FOL formula is a Gentzen formula and only horn for $n = 1$:

$\phi(C \sqsubseteq C_1 \sqcup \ldots \sqcup C_n) = \forall x.(C(x) \rightarrow \bigvee_{i \in [1,n]} C_i(x)) \Leftrightarrow \forall x.(C(x) \rightarrow \bigvee_{i \in [1,n]} C_i(x))$

In consequence of Theorem 4.5.1 on page 93, the formula cannot be translated into a set of Horn formulae.

□

Remark 4.5.4 *One could represent the Gentzen formula above in Disjunctive Datalog (Eiter et al., 1997), which allows disjunctions in the head of rules. We presented such an extension in (Motik et al., 2003). Due to the discouraging complexity results for Disjunctive Datalog (Dantsin et al., 2001), however, we do not meet our target of scalability with such an extension and we do not further regard this possibility here.*

Remark 4.5.5 *A direct consequence of corollary 4.5.7 is that we can **not** use disjunction in individual assertions, e.g. the following assertion cannot be translated to $\mathcal{LP}$.*

GOODTEACHER $\sqcup$ BADTEACHER(JOHANN-AMBROSIUS)

The following corollary follows directly from Corollary 4.5.7:

Corollary 4.5.8 (Equivalence $\sqcup$) *A disjunction $\sqcup$ with $n > 1$ disjuncts in class equivalence axioms can **not** be represented in $\mathcal{LP}$.*

Remark 4.5.6 ($\top$ and $\bot$ in disjunctions) *In consequence of the tautology $C \sqcup \top \equiv \top$ and corollary 4.5.2, we need $\mathcal{LP}_1$, i.e. equality, to represent the disjunction, if one disjunct is $\top$. Similarly, since $C \sqcup \bot \equiv C$, the results of corollaries 4.5.6, 4.5.7 and 4.5.8 still hold if more than two disjuncts exist and one disjunct is $\bot$. If two disjuncts are present, the disjunction can be represented in $\mathcal{LP}_0$ on the right-hand side of an inclusion axiom and consequently in class equivalence axioms.*

4.5.4.3. Negation

Negation in Description Logics corresponds to the classical form of negation in FOL. However, this form of negation is substantially different from the negation-as-failure that is available in some Logic Programming environments and logic databases.

Negation-as-failure implicitly applies the closed world assumption (CWA), viz. $\neg$COMPOSER(JOHANN-SEBASTIAN) holds, if we do not have a fact COMPOSER(JOHANN-SEBASTIAN). Classical negation does not make this conclusion, viz. absence of information does not allow to conclude the negation of information.

Corollary 4.5.9 (right-hand side $\neg$) *Negation $\neg B$ on the right-hand side of a DL inclusion axiom can be represented in $\mathcal{LP}_2$.*

Proof: The following transformations show that the corresponding formula is Horn but does not contain any positive literals:

- $B \neq \bot$
  $\phi(A \sqsubseteq \neg B) = \forall x.(A(x) \rightarrow \neg B(x)) \Leftrightarrow \forall x.(\neg A(x) \vee \neg B(x))$
- $B = \bot$
  $\neg\bot \equiv \top$, i.e. corollary 4.5.2 holds, which states that $\mathcal{LP}_1$ suffices.

□

Example 4.5.5 *A logic database would disallow any state of the knowledge base where the constraint introduced by negation is not met. Given the axiom* MAN $\sqsubseteq \neg$WOMAN *and the fact* MAN(JOHANN-AMBROSIUS)*, we will be prevented from making the assertion*

WOMAN(JOHANN-AMBROSIUS).

Corollary 4.5.10 (left-hand side ¬) *Negation $\neg A$ with $A \neq \bot$ on the left-hand side of a DL inclusion axiom can **not** be represented in $\mathcal{LP}$.*

Proof: The following transformation shows that the corresponding formula is always a Gentzen formula:

$\phi(\neg A \sqsubseteq B) = \forall x.(\neg A(x) \rightarrow B(x)) \Leftrightarrow \forall x.(A(x) \vee B(x))$

In consequence of Theorem 4.5.1 on page 93, the formula cannot be translated into a set of Horn formulae.

□

Remark 4.5.7 *As we will see in Section 5.4 on page 130, we can also not check for instances of some class $\neg A$, which is clearly possible in Description Logics.*

The following corollary follows directly from Corollary 4.5.10.

Corollary 4.5.11 (Equivalence ¬) *Negation $\neg B$ with $B \neq \bot$ in class equivalence axioms can **not** be represented in $\mathcal{LP}$.*

4.5.5. Enumeration

Classes can be defined by explicit enumeration of nominals (individuals with a well-known name) using the enumeration constructor. For example, a class CONTINENTS would include the nominals AFRICA, ASIA, EUROPE ....

The enumerated set of nominals is closed, viz. users cannot specify new individuals outside the definition of the enumeration. A naive translation of nominals $\{i_1, ..., i_n\}$ which would translate each enumeration into a new class name N_i, along with membership assertions $N_i(i_j)$ for each nominal, is therefore not correct, since the semantics of enumeration specifies that newly asserted individuals must be equivalent to one of the nominals used in the definition of the class.

4.5.5.1. Enumeration with $n > 1$ elements

Corollary 4.5.12 (left-hand side Enumeration $\{i_1, \ldots, i_n\}$) *Enumeration $\{i_1, \ldots, i_n\}$ with $n > 1$ on the left-hand side of a DL inclusion axiom can be represented in $\mathcal{LP}_1$.*

Proof: The following transformation shows that the corresponding formula is a conjunction of Horn formulae:

$\phi(\{i_1, i_2, \ldots i_n\} \sqsubseteq C) = \forall x.((x = i_1 \vee \ldots \vee x = i_n) \rightarrow C(x))$
$\Leftrightarrow \forall x.(\bigwedge_{l \in [1,n]} x = i_l \rightarrow C(x))$

□

Corollary 4.5.13 (right-hand side Enumeration $\{i_1, \ldots, i_n\}$) *Enumeration $\{i_1, \ldots, i_n\}$ on the right-hand side of a DL inclusion axiom can **not** be represented in $\mathcal{LP}$.*

Proof: The translation yields a Gentzen formula:

$\phi(C \sqsubseteq \{i_1, i_2, \ldots i_n\}) = \forall x.(\neg C(x) \vee x = i_1 \vee \ldots \vee x = i_n)$

In consequence of Theorem 4.5.1 on page 93, the formula cannot be translated into a set of Horn formulae.

□

Remark 4.5.8 *Obviously, these rules can be optimized in an implementation into n assertions $C(i_n)$. We do not directly incorporate this optimization here, since it would complicate the translation function presented in Section 5.2 on page 119.*

The following corollary follows directly from Corollary 4.5.13:

Corollary 4.5.14 (Equivalence $\{i_1, \ldots, i_n\}$) *Enumeration $\{i_1, \ldots, i_n\}$ in class equivalence axioms can **not** be represented in $\mathcal{LP}$.*

4.5.5.2. Enumeration with $n = 1$ element

Corollary 4.5.15 (Equivalence $\{i_1\}$) *Enumeration $\{i_1\}$ in class equivalence axioms can be represented in $\mathcal{LP}_1$.*

Proof: As Corollary 4.5.12 shows the possibility to translate the left-hand side of inclusion axioms, it remains to show the right-hand side, which is a single Horn formula $\phi(C \sqsubseteq \{i_1\}) = \forall x.(C(x) \rightarrow (x = i_1))$.

□

4.5.5.3. hasValue

The results for enumeration with one element can be used for the hasValue construct, which is available in OWL as a convenient primitive for stating an existential restriction that limits the value of some property to a single nominal $\exists R.\{a\}$. The resulting rules can be simplified such that equality is absorbed in FOL. Therefore hasValue can be represented in $\mathcal{LP}_0$.

Corollary 4.5.16 (Equivalence hasValue $\exists R.\{a\}$) *hasValue $\exists R.\{a\}$ in class equivalence axioms can be represented in $\mathcal{LP}_0$.*

Proof: The following transformations show that we obtain a conjunction of Horn formulae:

- hasValue on the left-hand side of DL inclusion axioms

  $\phi(\exists R.\{a\} \sqsubseteq C) = \forall x.((\exists y.(R(x,y) \wedge (y = a)) \rightarrow C(x)))$
  $\Leftrightarrow \forall x.(R(x,a) \rightarrow C(x))$

- hasValue on the right-hand side of DL inclusion axioms

  $\phi(C \sqsubseteq \exists R.\{a\}) = \forall x.(C(x) \rightarrow \exists y.(R(x,y) \wedge (y = a)))$
  $\Leftrightarrow \forall x.(C(x) \rightarrow R(x,a))$

- Class equivalence axioms

  $\phi(C \equiv \exists R.\{a\}) = \phi(\exists R.\{a\} \sqsubseteq C) \wedge \phi(C \sqsubseteq \exists R.\{a\})$
  $\Leftrightarrow \forall x.((R(x,a) \rightarrow C(x)) \wedge (C(x) \rightarrow R(x,a)))$

□

4.5.6. Property Restrictions

4.5.6.1. Value Restriction

Remark 4.5.9 *Preprocessing transforms $\forall R.\top$ into $\top$. The translation then follows from Corollary 4.5.2 on page 96.*

Corollary 4.5.17 (left-hand side $\forall R.D$) *Value restrictions $\forall R.D$ for $D \neq \{\top\}$ on the left-hand side of a DL inclusion axiom can **not** be represented in $\mathcal{LP}$.*

Proof: The following transformation shows that we obtain one conjunct that is not Horn:

$\phi(\forall R.D \sqsubseteq C) = \forall x.((\forall y.(R(x,y) \rightarrow D(y))) \rightarrow C(x))$
$\Leftrightarrow \forall x.(\neg(\forall y.(\neg R(x,y) \vee D(y))) \vee C(x))$
$\Leftrightarrow \forall x.\exists y((R(x,y) \wedge \neg D(y)) \vee C(x))$
$\Leftrightarrow \forall x.\exists y.((R(x,y) \vee C(x)) \wedge (D(y) \rightarrow C(x)))$

In consequence of Theorem 4.5.1 on page 93, the formula cannot be translated into a set of Horn formulae.

□

The following corollary follows directly from Corollary 4.5.17:

Corollary 4.5.18 (Equivalence $\forall R.D$) *Value Restrictions $\forall R.D$ for $D \neq \{\top, \bot\}$ in class equivalence axioms can **not** be represented in $\mathcal{LP}$.*

Corollary 4.5.19 (right-hand side $\forall R.D$) *Value restrictions $\forall R.D$ on the right-hand side of a DL inclusion axiom can be represented in*

- *$\mathcal{LP}_0$ for $D \neq \{\top, \bot\}$*
- *$\mathcal{LP}_2$ for $D = \bot$.*

Proof:

- For $D \neq \{\top, \bot\}$, the following transformation shows that the corresponding formula is a Horn formula:

  $\phi(C \sqsubseteq \forall R.D) = \forall x.(C(x) \rightarrow (\forall y.(R(x,y) \rightarrow D(y))))$
  $\Leftrightarrow \forall x.(\neg C(x) \vee (\forall y.(\neg R(x,y) \vee D(y))))$
  $\Leftrightarrow \forall x,y.(\neg C(x) \vee \neg R(x,y) \vee D(y)) \Leftrightarrow \forall x,y.((C(x) \wedge R(x,y)) \rightarrow D(y))$

- For $D = \bot$, the following transformation shows that the formula is Horn and does not have any positive literals, which requires $\mathcal{LP}_2$:

  $\phi(C \sqsubseteq \forall R.\bot) = \forall x.(C(x) \rightarrow (\forall y.(R(x,y) \rightarrow \neg(y = y))))$
  $\Leftrightarrow \forall x.(\neg C(x) \vee (\forall y.(\neg R(x,y) \vee \neg(y = y))))$
  $\Leftrightarrow \forall x,y.(\neg C(x) \vee \neg R(x,y) \vee \neg(y = y))$

□

4.5.6.2. Existential Restriction

Remark 4.5.10 *Preprocessing simplifies $\exists R.\bot$ to $\bot$. The translation then immediately follows from Corollaries 4.5.3 and 4.5.4.*

Corollary 4.5.20 (left-hand side $\exists R.C$) *Existential restrictions $\exists R.C$ with $C \neq \bot$ on the left-hand side of a DL inclusion axiom can be represented in $\mathcal{LP}_0$.*

Proof: The following transformation shows that the corresponding formula is a Horn formula:

$\phi(\exists R.C \sqsubseteq D) = \forall x.((\exists y.(R(x,y) \wedge C(y))) \rightarrow D(x)) \Leftrightarrow \forall x,y.((R(x,y) \wedge C(y)) \rightarrow D(x))$

□

Corollary 4.5.21 (right-hand side $\exists R.C$) *Existential restrictions $\exists R.C$ with $C \neq \bot$ on the right-hand side of a DL inclusion axiom can be represented in $\mathcal{LP}_3$.*

Proof: The following transformations show that the corresponding formula is a conjunction of Horn formulae. We require, however, skolemization to replace the existential quantifier which leads to equi-satisfiability of both formulae:

$\phi(D \sqsubseteq \exists R.C) = \forall x.(D(x) \rightarrow (\exists y.(R(x,y) \wedge C(y))))$
$\Leftrightarrow \forall x.(\neg D(x) \vee (\exists y.(R(x,y) \wedge C(y))))$
$\Leftrightarrow \forall x.\exists y.((D(x) \rightarrow R(x,y)) \wedge (D(x) \rightarrow C(y)))$

Skolemization: Replace $\exists y$ with new, previously unused $f(x)$:

$\forall x.((D(x) \rightarrow R(x, f(x))) \wedge (D(x) \rightarrow C(f(x))))$

□

Example 4.5.6 (Non-termination) *In case of terminological cycles, we can easily effect non-terminating rules. For example, the axiom $C \sqsubseteq \exists R.C$ and some individual assertion $C(a)$ yields infinite recursion, since we would have to generate*

$$\{C(f(a)), C(f(f(a))), C(f(f(f(a)))), \ldots, C(f^{\infty}(a))\}$$

Remark 4.5.11 *Logically, this infinite recursion does not pose any difficulties. However, in practise, the infinite recursion is not acceptable. In fact, the same infinite recursion would also occur in tableau based reasoners, if the blocking technique (Buchheit et al., 1993) would not be used. In a nutshell, blocking regains termination by trying to detect such cyclic computations and then blocking the application of generating rules.*

The detection of infinite recursion has to be provided by the logic database. For example, Euler (Roo, 2002) prevents this recursion by so-called Euler paths, which avoid that the same terms are unified twice.

The following corollary follows directly from Corollaries 4.5.20 and 4.5.21.

Corollary 4.5.22 (Equivalence $\exists R.C$) *Existential Restrictions $\exists R.C$ with $C \neq \bot$ in class equivalence axioms can be represented in a $\mathcal{LP}$ that allows function symbols.*

4.5.6.3. Minimal Number Restrictions

Remark 4.5.12 *The simplification step carried out during preprocessing uses the equivalences $\geq 0\, R \equiv \top$ and $\geq 1\, R \equiv \exists R.\top$ to eliminate minimal number restrictions $\geq 0\, R$ and $\geq 1\, R$. The translation of these constructors therefore follows from the translation of $\top$ and $\exists R.\top$ respectively.*

Corollary 4.5.23 (left-hand side $\geq n\, R$) *The minimal number restriction $\geq n\, R$ with $n > 1$ on the left-hand side of a DL inclusion axiom can **not** be represented in $\mathcal{LP}$.*

Proof: The following transformation shows that the formula is equivalent to a single Gentzen formula:

$\phi(\geq n\, R \sqsubseteq C) =$
$\forall x.((\exists y_1, \ldots, y_n. \bigwedge_{i \neq j} \neg(y_i = y_j) \bigwedge_i R(x, y_i)) \rightarrow C(x)) \Leftrightarrow$
$\forall x, y_1, \ldots, y_n.((\bigvee_{i \neq j}(y_i = y_j) \bigvee_i \neg R(x, y_i)) \vee C(x))$

In consequence of Theorem 4.5.1 on page 93, the formula cannot be translated into a set of Horn formulae.

□

Corollary 4.5.24 (right-hand side $\geq n\, R$) *The minimal number restriction $\geq n\, R$ with $n > 1$ on the right-hand side of a DL inclusion axiom can be represented in $\mathcal{LP}_3$.*

Proof: The following transformation shows that we can get a conjunction of Horn formulae:

$\phi(C \sqsubseteq \geq n\, R) =$
$\forall x.(C(x) \rightarrow (\exists y_1, \ldots, y_n. \bigwedge_{i \neq j} \neg(y_i = y_j) \bigwedge_i R(x, y_i))$
$\Leftrightarrow \forall x. \exists y_1, \ldots, y_n.(\neg C(x) \vee (\bigwedge_{i \neq j} \neg(y_i = y_j) \bigwedge_i R(x, y_i))$
$\Leftrightarrow \forall x. \exists y_1, \ldots, y_n.(\bigwedge_{i \neq j}(\neg C(x) \vee \neg(y_i = y_j)) \wedge \bigwedge_i (\neg C(x) \vee R(x, y_i)))$

We have to introduce n fresh skolem functions of arity 1 to eliminate the existential quantifier and carry out substitutions for all y_i :$\{[y_i/f_i(x)]\}$. This allows to obtain the following equi-satisfiable formula:

$\forall x.(\bigwedge_{1\leq i<j\leq n}(\neg C(x) \vee \neg(f_i(x) = f_j(x))) \wedge \bigwedge_{i\in[1,n]}(C(x)\rightarrow R(x, f_i(x))))$

This is a large conjunction of Horn formulae, some of which make use of function symbols and have no positive literals, i.e. they are integrity constraints.

□

The following corollary immediately follows from Corollary 4.5.23:

Corollary 4.5.25 (Equivalence $\geq n\, R$) *The minimal number restriction $\geq n\, R$ with $n > 1$ in class equivalence axioms can **not** be represented in $\mathcal{LP}$.*

4.5.6.4. Maximum Number Restriction

Remark 4.5.13 *The simplification step carried out during preprocessing uses the equivalences $\leq 0\, R \equiv \forall R.\bot$ to eliminate maximal number restrictions $\leq 0\, R$. The translation of $\leq 0\, R$ therefore immediately follows from the translation of $\forall R.\bot$.*

Corollary 4.5.26 (right-hand side $\leq 1\, R$) *The maximal number restriction $\leq n\, R$ with $n = 1$ on the right-hand side of a DL inclusion axiom can be represented in $\mathcal{LP}_1$.*

Proof: The following transformation shows that the corresponding formula is Horn and requires equality in the head:

$\phi(D \sqsubseteq \leq 1\, R) = \forall x.(D(x)\rightarrow(\forall y_1, y_2(R(x, y_1) \wedge R(x, y_2)\rightarrow y_1 = y_2)))$
$\Leftrightarrow \forall x, y_1, y_2.(\neg D(x) \vee \neg R(x, y_1) \vee \neg R(x, y_2) \vee y_1 = y_2)$

□

Corollary 4.5.27 (Translation of $\leq n\, R$) *The maximal number restriction $\leq n\, R$ with $n > 0$ can **not** be represented in $\mathcal{LP}$ when used*

- *on the left-hand side of inclusion axioms;*
- *on the right-hand side of inclusion axioms (for $n > 1$);*
- *in class equivalence axioms.*

Proof:

- $\leq n\, R$ on the left-hand side of DL inclusion axioms

  Using the equivalences $\leq n\, R \equiv \neg{\geq}\,(n+1)\, R$ and $\neg C \sqsubseteq \neg D \Leftrightarrow D \sqsubseteq C$ we can transform the $\leq n\, R \sqsubseteq C$ into $\neg C \sqsubseteq {\geq}\,(n+1)\, R$

  $\phi(\neg C \sqsubseteq {\geq}\,(n+1)\, R) = \forall x.(\neg C(x) \rightarrow (\exists y_1, \ldots, y_{(n+1)}.\bigwedge_{i \neq j} \neg(y_i = y_j) \bigwedge_i R(x, y_i))$
  $\Leftrightarrow \forall x.\exists y_1, \ldots, y_{(n+1)}.(C(x) \vee (\bigwedge_{i \neq j} \neg(y_i = y_j) \bigwedge_i R(x, y_i))$
  $\forall x.\exists y_1, \ldots, y_n.(\bigwedge_{i \neq j}(C(x) \vee \neg(y_i = y_j)) \wedge \bigwedge_i (C(x) \vee R(x, y_i)))$

- $\leq n\, R$ on the right-hand side of DL inclusion axioms

  Using the equivalences $\leq n\, R \equiv \neg{\geq}\,(n+1)\, R$ and $\neg C \sqsubseteq \neg D \Leftrightarrow D \sqsubseteq C$, we can transform the $C \sqsubseteq \leq n\, R$ into ${\geq}\,(n+1)\, R \sqsubseteq \neg C$. Since $n > 1$ Corollary 4.5.23 applies, which states that the constructor cannot be translated.

- Class equivalence axioms

  $\phi(C \equiv \leq n\, R) = \phi(\leq n\, R \sqsubseteq C) \wedge \phi(C \sqsubseteq \leq n\, R)$

In all three cases, Theorem 4.5.1 on page 93 applies and states that the formula cannot be translated into a set of Horn formulae.

□

Remark 4.5.14 *The inability to translate arbitrary number restrictions is not tragic, since these language features are seldomly used in practise (Goble, 2000) (cf. also Section 5.3.2 on page 128 for the analysis of the DAML.ORG ontology library).*

4.5.6.5. Exact Number Restriction

OWL DL additionally includes a convenience primitive `cardinality`, which allows to state exact number restrictions. It can be understood as the conjunction of stating both a maximum number restriction and a minimum number restriction with the same number on the same property.

We can directly deduce from the previous corollaries, that the minimum and maximum cardinalities may only appear on the same side of an inclusion axiom for $n \in \{0, 1\}$. For $n > 1$, Corollary 4.5.27 states that $\leq n\, R$ can only appear on the left-hand side of an inclusion axiom, whereas Corollary 4.5.24 states that $\geq n\, R$ can only appear on the right-hand side of an inclusion axiom.

Corollary 4.5.28 (right-hand side $= 0\, R$) *The maximal number restriction $= 0\, R$ on the right-hand side of DL inclusion axioms can be represented in $\mathcal{LP}_2$.*

Proof: $C \sqsubseteq =0\, R \Leftrightarrow C \sqsubseteq \leq 0\, R \sqcap \geq 0\, R \Leftrightarrow C \sqsubseteq \forall R.\bot$.
The translation directly follows from Corollary 4.5.19.

□

Corollary 4.5.29 (right-hand side $=1\, R$) *The maximal number restriction $=1\, R$ on the right-hand side of an DL inclusion axiom can be represented in $\mathcal{LP}_3$.*

Proof: $C \sqsubseteq =1\, R \Leftrightarrow C \sqsubseteq \geq 1\, R \sqcap \leq 1\, R \Leftrightarrow C \sqsubseteq \exists R.\top \sqcap \leq 1\, R$.
The translation directly follows from corollaries 4.5.21 and 4.5.26.

□

Example 4.5.7 ($=1\, R$) *Using the transformations carried out in the proofs of Corollaries 4.5.21, 4.5.26 and 4.5.29, we can now translate the inclusion axiom*

$$\text{HUMAN} \sqsubseteq =1\, \text{HASLIFE}$$

into the following set of LP rules:

$$y_1 = y_2 \texttt{:-} \text{HUMAN}(x), \text{HASLIFE}(x, y_1), \text{HASLIFE}(x, y_2).$$
$$f_1(x) = f_1(x) \texttt{:-} \text{HUMAN}(x).$$
$$\text{HASLIFE}(x, f_1(x)) \texttt{:-} \text{HUMAN}(x).$$

4.5.7. Property Characteristics

OWL DL includes various possibilities to augment the definition of properties. Most of these definitions, e.g. domain, range and functionalities, can be understood as syntactic shortcuts for other expressions (cf. Table 3.2 on page 64). We can therefore reuse the established results and directly present the result of the transformation. The reader may note that we eliminated the tautological conjunct $x = x$ in the transformation of translated global domain and range restrictions:

$$\phi_x(\top \sqsubseteq \forall P^{-}.C_1) = \forall x, y.((P(x, y) \wedge (y = y)) \rightarrow C_1(x)) \Leftrightarrow \forall x, y.(P(x, y) \rightarrow C_1(x))$$

Inverse and transitivity of properties are indeed new features in $\mathcal{L}_{In}$. However, their FOL equivalent is a Horn formula (cf. Table 4.4 on page 90). Therefore, no transformation has to be carried out and the corresponding LP rule follows immediately. The resulting LP translations for property characteristics are summarized in Table 4.6 on page 110.

OWL DL Abstract Syntax	DL	LP
`ObjectProperty` (P		
domain(C_1)	$\top \sqsubseteq \forall P^-.C_1$	$C_1(x)$:- $P(x,y)$.
...	...	...
domain(C_n)	$\top \sqsubseteq \forall P^-.C_n$	$C_n(x)$:- $P(x,y)$.
range(R_1)	$\top \sqsubseteq \forall P.R_1$	$R_1(y)$:- $P(x,y)$.
...	...	...
range(R_n)	$\top \sqsubseteq \forall P.R_n$	$R_n(y)$:- $P(x,y)$.
inverseOf(Q)	$P \equiv Q^-$	$P(x,y)$:- $Q(y,x)$.
		$Q(x,y)$:- $P(y,x)$.
Functional	$\top \sqsubseteq \leq 1\, P$	$y_1 = y_2$:- $P(x,y_1), P(x,y_2)$.
InverseFunctional	$\top \sqsubseteq \leq 1\, P^-$	$x_1 = x_2$:- $P(x_1,y), P(x_2,y)$.
Symmetric	$P \equiv P^-$	$P(x,y)$:- $P(y,x)$.
Transitive	$P^+ \sqsubseteq P$	$P(x,z)$:- $P(x,y), P(y,z)$.
)		

Table 4.6.: Translation of Property Characteristics to LP

4.5.8. Summary

Table 4.7 on page 111 summarizes the previous corollaries and shows which language feature can be translated fully *(X)* or may only appear on the left- *(L)* respectively right-hand side *(R)* of inclusion axioms or cannot be translated at all *(-)*.

4.6. Conclusion

4.6.1. Contribution

In this chapter we showed how a meaning-preserving translation of some DL axioms to a logic program can be achieved via a translation into FOL formulae and subsequent transformation into a logic program. Moreover, we presented a novel technique for preprocessing DL axioms such that an effective translation into LP is possible. Preprocessing removes redundancies in the syntax and removes negation from TBox axioms (as far as possible). We showed which significant and expressive fragment of the $\mathcal{SHINO}$ DL can be translated into Logic Programs and illustrated the necessary transformation steps to gain LP rules.

We use this fragment in the next chapter for the definition of a family of new Description Logics $\mathcal{L}_i$ and present a semantics for these languages, which is based on a formal translation into logic programs. Moreover, we show how prototypical Description Logic reasoning problems can be reduced to answering queries on Logic Programs and study the expressiveness and the complexity of the new languages.

$\mathcal{SHINO}$ Primitive	DL	Datalog $\mathcal{LP}_0$	Datalog$^{=}$ $\mathcal{LP}_1$	Datalog$^{=,IC}$ $\mathcal{LP}_2$	Prolog$^{=,IC}$ $\mathcal{LP}_3$
ABox					
Indiv. assertion	$C(a)$	X	X	X	X
Prop. assertion	$P(a,b)$	X	X	X	X
Indiv. equivalence	$a = b$	-	X	X	X
Indiv. inequivalence	$\neg(a = b)$	-	-	X	X
TBox					
Atomic Class	A	L,R	L,R	L,R	L,R
Atomic Properties	P	L,R	L,R	L,R	L,R
Top	$\top$	-	L,R	L,R	L,R
Bottom	$\bot$	-	-	R	R
Conjunction	$\sqcap$	L,R	L,R	L,R	L,R
Disjunction	$\sqcup$	L	L	L	L
Atomic Negation	$\neg A$	-	-	R	R
Value Restriction	$\forall R.(C \neq \bot)$	R	R	R	R
Value Restriction	$\forall R.\bot$	-	-	R	R
Existential Restriction	$\exists R.(C \neq \bot)$	L	L	L	L,R$^{(1)}$
Existential Restriction	$\exists R.\bot$	-	-	R	R
One-Of 1	$\{o\}$	-	L,R	L,R	L,R
hasValue	$\exists R.\{o\}$	L,R	L,R	L,R	L,R
One-Of	$\{o_1, \ldots, o_n\}$	-	L	L	L
Min. Cardinality 0	$\geq 0\, R = \top$	-	L,R	L,R	L,R
Min. Cardinality 1	$\geq 1\, R = \exists R.\top$	-	L	L	L,R$^{(1)}$
Min. Cardinality n	$\geq n\, R$	-	-	-	R
Max. Cardinality 0	$\leq 0\, R = \forall R.\bot$	-	-	R	R
Max. Cardinality 1	$\leq 1\, R$	-	R	R	R
Max. Cardinality n	$\leq n\, R$	-	-	-	-
Property Characteristics					
Domain		X	X	X	X
Range		X	X	X	X
Inverse		X	X	X	X
Symmetry		X	X	X	X
Transitivity		X	X	X	X
Functionality		-	X	X	X
InverseFunctionality		-	X	X	X

(1) Only satisfiability preserving (due to skolemization).
(L) = May appear in body of a LP rule, left-hand side of a DL inclusion axiom.
(R) = May appear in head of a LP rule, right-hand side of a DL inclusion axiom.

Table 4.7.: Expressivity of Different LP Variants

In particular, we show that most parts of currently available Web ontologies can be expressed in the new languages and that the family of languages is tractable.

4.6.2. Further uses

Other work can make use of the translation studied in this chapter. For example, one can:

- *"Build rules on top of ontologies"*: It enables the logic program to have access to ontological definitions of the vocabulary (e.g., classes, properties and individuals) used by the rules.
- *"Build ontologies on top of rules"*: Conversely, the technique enables ontological definitions to operate on the output of rules, which basically create the ABox upon which the TBox operates.
- *"Learn ontologies"*: It allows to apply Inductive Logic Programming (ILP) techniques to learn ontologies from a given set of relational data by reversing the transformation and translation. This would allow to transfer the output of ILP algorithms into Description Logic axioms.
- *"Extend the translation"*: It can serve as the starting point (Motik et al., 2003) for extending the translation into more expressive Logic Programming variants such as Disjunctive Datalog (Eiter et al., 1997), which allows disjunctions in the head of rules.

5. Description Logic Programs

"If our basic tool, the language in which we design and code our programs, is also complicated, the language itself becomes part of the problem rather than part of its solution."
C. A. R. Hoare
The Emperor's Old Clothes (Turing Award Lecture)

This chapter defines Description Logic Programs (DLP) as a family of new Description Logics $\mathcal{L}_i$ and presents a semantics for these languages which is based on a translation into Logic Programs. It thereby utilizes the results established in the previous chapter to extend the $\mathcal{L}_0$ language, which was presented in (Grosof et al., 2003). Moreover, the chapter shows how prototypical Description Logic reasoning problems can be reduced to answering queries on Logic Programs and studies the expressiveness and the complexity of the new languages. In particular, we show that most parts of currently available Web ontologies can be expressed in the new languages and that the family of languages is tractable.

The chapter is organized as follows: Section 5.1 introduces the syntax of the individual languages $\mathcal{L}_i$ in the DLP family. Section 5.2 on page 119 presents the semantics of the DLP languages, which is based on a formal translation of DLP knowledge bases into extended Logic Programs. Section 5.3 on page 125 describes the expressivity of the individual $\mathcal{L}_i$ languages through a comparison with the different proposals for Web ontology languages in the Semantic Web, i.e. RDFS, OWL Lite and OWL DL. We particularly show that the majority of currently available Web ontologies can be expressed in DLP. Section 5.4 on page 130 shows how typical DL reasoning problems can be reduced to answering queries on logic programs. We also show which reasoning problems make sense for a particular $\mathcal{L}_i$ variant. Section 5.5 on page 141 briefly characterizes the complexity of reasoning with each DLP variant and the translation process itself and shows a more tight polynomial bound for the Datalog-based DLP languages. Section 5.6 on page 143 summarizes the contribution of this chapter and links to the last chapter in this part.

5.1. Syntax

The results of chapter 4 naturally lead to the definition of a set of paired DL languages, which allow to construct class descriptions on a certain side of an inclusion axiom. We will refer to these paired languages as $\mathcal{L}_i^L$ and $\mathcal{L}_i^R$ respectively.

$\mathcal{L}_i^R$ is the language which allows to form class descriptions on the right-hand side of DL inclusion axioms. Conversely, $\mathcal{L}_i^L$ is the language which allows to form classes on the left-hand side of DL class inclusion axioms.

$\mathcal{L}_i^R$ and $\mathcal{L}_i^L$ are extensions of a core language $\mathcal{L}_i$. $\mathcal{L}_i$ captures class descriptions that can be expressed on both sides of inclusion axioms and, consequently, is the language that can be used to formulate class equivalence axioms. We will be able to ensure by construction that all inclusion axioms formed with $\mathcal{L}_i^R$ and $\mathcal{L}_i^L$ and all equivalence axioms formed with $\mathcal{L}_i$ can be translated into the Logic Programming variant $\mathcal{LP}_i$.

We can recall from Section 4.2.3 on page 83 that we consider the following variants of Logic Programming systems as possible target logics:

$\mathcal{LP}_0$ Datalog,

$\mathcal{LP}_1$ Datalog(=), which augments Datalog with equality,

$\mathcal{LP}_2$ Datalog(=,IC), which augments Datalog(=) with integrity constraints,

$\mathcal{LP}_3$ Prolog(=,IC), which augments Datalog(=,IC) with function symbols and allows unsafe rules.

The reader may note that all $\mathcal{L}_i$ include language constructors that can later be eliminated in the preprocessing step described in Section 4.3 on page 83. We include these constructors to allow an epistemologically adequate representation of knowledge.

In the following, we use the established notation, viz. A refers to an atomic class, C and D to class descriptions, P and R to atomic properties and o is an individual. Class descriptions can be built inductively from atomic class names using the class constructors available in a given language $\mathcal{L}_i^{L/R}$. Figure 5.1 illustrates the constructors provided by the DLP variants.

5.1.1. Datalog $(\mathcal{L}_0)$

$\mathcal{L}_0$ presents the DL language which allows the construction of class equivalence axioms that can be translated into Datalog programs. The extensions $\mathcal{L}_0^L$ and $\mathcal{L}_0^R$

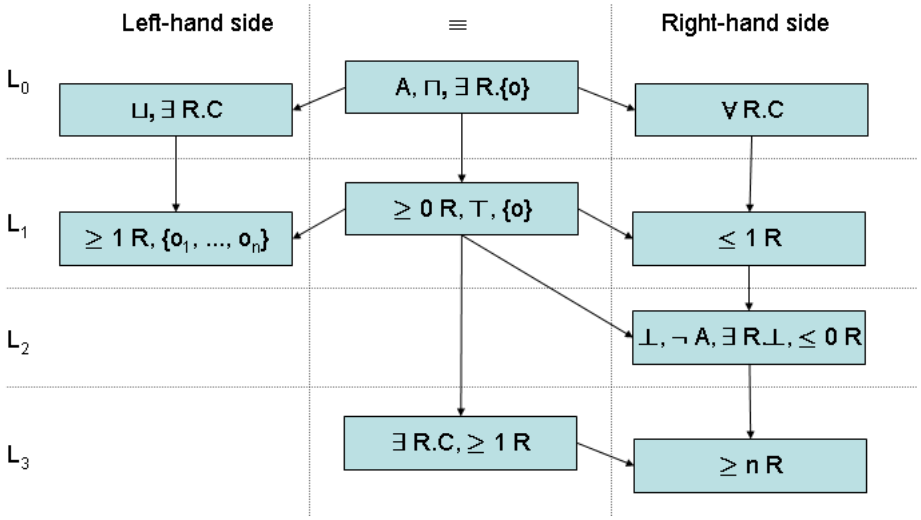


Figure 5.1.: DLP Language Variants

allow to formulate class descriptions that can appear on the left ($\mathcal{L}_0^L$) or right ($\mathcal{L}_0^R$) hand side of DL class inclusion axioms.

Definition 5.1.1 ($\mathcal{L}_0$) *Class descriptions in $\mathcal{L}_0$ are formed according to the following syntax rules:*

C, D	$\rightarrow$	A	|	*(atomic class)*
		$C \sqcap D$	|	*(conjunction)*
		$\exists R.\{o\}$	|	*(hasValue)*

Definition 5.1.2 ($\mathcal{L}_0^R$) *Class descriptions in $\mathcal{L}_0^R$ are formed according to the syntax rules of $\mathcal{L}_0$ and the additional rule:*

C, D	$\rightarrow$	$\forall R.C$	|	*(value restriction)*

Definition 5.1.3 ($\mathcal{L}_0^L$) *Class descriptions in $\mathcal{L}_0^L$ are formed according to the syntax rules of $\mathcal{L}_0$ and the additional rules:*

C, D	$\rightarrow$	$C \sqcup D$	|	*(disjunction)*
		$\exists R.C$	|	*(existential restriction)*

Definition 5.1.4 ($\mathcal{T}_0^{DLP}$) *A DLP TBox $\mathcal{T}_0^{DLP}$ is a set DLP TBox axioms (cf. Table 5.1) where C is a $\mathcal{L}_0^L$ class, D is a $\mathcal{L}_0^R$ class, E, F are $\mathcal{L}_0$ classes and P, Q are properties.*

$C \sqsubseteq D$,	(class inclusion)
$E \equiv F$,	(class equivalence)
$P \sqsubseteq Q$,	(property subsumption)
$P \equiv Q$,	(property equivalence)
$P \equiv Q^-$,	(property inverse)
$P \equiv P^-$,	(property symmetry)
$P^+ \sqsubseteq P$,	(property transitivity)
$\top \sqsubseteq \forall P^-.D$,	(property domain)
$\top \sqsubseteq \forall P.D$,	(property range)

Table 5.1.: DLP TBox axioms

Remark 5.1.1 *We allow users to state property domains and ranges, since $\top$ can be eliminated from the resulting rules. For example, the axiom $\top \sqsubseteq \forall P^-.D$ will be translated to the rule $\{D(x) \texttt{:-} P(x,y).\}$ instead of $\{D(x) \texttt{:-} (x = x), P(x,y).\}$, which is clearly equivalent.*

Definition 5.1.5 ($\mathcal{A}_0^{DLP}$) *A DLP ABox $\mathcal{A}_0^{DLP}$ is a set of axioms of the form*

$P(a,b)$	*(property fillers)*
$D(a)$	*(individual assertion)*

where P is a property and D is a $\mathcal{L}_0^R$ class and a, b are individuals.

Definition 5.1.6 ($\mathcal{KB}_0^{DLP}$) *A DLP knowledge base $\mathcal{KB}_0^{DLP}$ is a pair $\langle \mathcal{T}_0^{DLP}, \mathcal{A}_0^{DLP} \rangle$.*

5.1.2. Datalog with equality ($\mathcal{L}_1$)

$\mathcal{L}_1$ presents a DL language which allows the definition of class equivalence axioms that can be translated into Datalog(=) programs. The extensions $\mathcal{L}_1^L$ and $\mathcal{L}_1^R$ allow to formulate class descriptions that can appear on the left ($\mathcal{L}_1^L$) or right ($\mathcal{L}_1^R$) hand side of DL class inclusion axioms.

Definition 5.1.7 ($\mathcal{L}_1$) *Class descriptions in $\mathcal{L}_1$ are formed according to the syntax rules of $\mathcal{L}_0$ and the additional rules:*

C, D	$\rightarrow$	$\{o\}$	|	*(one-of with arity 1)*
		$\top$	|	*(Universal class)*
		$\geq 0\, R$	|	*(Min. Cardinality 0)*

Definition 5.1.8 ($\mathcal{L}_1^R$) *Class descriptions in $\mathcal{L}_1^R$ are formed according to the syntax rules of $\mathcal{L}_1$ and $\mathcal{L}_0^R$ and the additional rule:*

C, D	$\rightarrow$	$\leq 1\, R$	|	*(Max. Cardinality 1)*

Definition 5.1.9 ($\mathcal{L}_1^L$) *Class descriptions in $\mathcal{L}_1^L$ are formed according to the syntax rules of $\mathcal{L}_1$ and $\mathcal{L}_0^L$ and the additional rules:*

C, D	$\rightarrow$	$\geq 1\, R$	|	*(Min. Cardinality 1)*
		$\{o_1, \ldots, o_n\}$	|	*(one-of with arity n)*

Definition 5.1.10 ($\mathcal{T}_1^{DLP}$) *A DLP TBox $\mathcal{T}_1^{DLP}$ is a set of DLP TBox axioms (cf. Table 5.1 on page 116) where each C is a $\mathcal{L}_1^L$ class, D is a $\mathcal{L}_1^R$ class, E, F are $\mathcal{L}_1$ classes and P, Q are properties.*

Definition 5.1.11 ($\mathcal{A}_1^{DLP}$) *A DLP ABox $\mathcal{A}_1^{DLP}$ is a set of $\mathcal{A}_0^{DLP}$ axioms and a set of axioms of the form*

$D(a)$	*(individual assertion)*
$a = b$	*(individual equivalence)*

where D is a $\mathcal{L}_1^R$ class and a, b are individuals.

Definition 5.1.12 ($\mathcal{KB}_1^{DLP}$) *A DLP knowledge base $\mathcal{KB}_1^{DLP}$ is a pair $\langle \mathcal{T}_1^{DLP}, \mathcal{A}_1^{DLP} \rangle$.*

5.1.3. Datalog with equality and integrity constraints $(\mathcal{L}_2)$

$\mathcal{L}_2$ presents a DL language which allows the construction of class equivalence axioms that can be translated into Datalog(=,IC) programs. The extensions $\mathcal{L}_2^L$ and $\mathcal{L}_2^R$ allow to formulate class descriptions that can appear on the left ($\mathcal{L}_2^L$) or right ($\mathcal{L}_2^R$) hand side of DL class inclusion axioms.

Definition 5.1.13 ($\mathcal{L}_2$) *Class descriptions in $\mathcal{L}_2$ are formed according to the syntax rules of $\mathcal{L}_1$.*

Definition 5.1.14 ($\mathcal{L}_2^R$) *Class descriptions in $\mathcal{L}_2^R$ are formed according to the syntax rules of $\mathcal{L}_1^R$ and the following additional rules:*

C, D	$\rightarrow$	$\neg A$	|	*(atomic negation)*
		$\bot$	|	*(empty class)*
		$\leq 0\, R$	|	*(Max. Cardinality 0)*
		$\exists R.\bot$	|	*(Prop. has filler that belongs to $\bot$)*

Remark 5.1.2 *The surface syntax exposed to users can actually be richer with respect to negation, viz. users may be allowed to state non-atomic negation in front of class description, as long as the expression can be normalized into a syntactically valid $\mathcal{L}_2^R$ expression.*

Definition 5.1.15 ($\mathcal{L}_2^L$) *Class descriptions in $\mathcal{L}_2^L$ are formed according to the syntax rules of $\mathcal{L}_1^L$.*

Definition 5.1.16 ($\mathcal{T}_2^{DLP}$) *A DLP TBox $\mathcal{T}_2^{DLP}$ is a set of DLP TBox axioms (cf. Table 5.1 on page 116) where each C is a $\mathcal{L}_2^L$ class, each D is a $\mathcal{L}_2^R$ class, E, F are $\mathcal{L}_2$ classes and P, Q are properties.*

Definition 5.1.17 ($\mathcal{A}_2^{DLP}$) *A DLP ABox $\mathcal{A}_2^{DLP}$ is a set of $\mathcal{A}_1^{DLP}$ axioms and a set of axioms of the form*

$D(a)$	*(individual assertion)*
$\neg(a = b)$	*(individual inequivalence)*

where D is a $\mathcal{L}_2^R$ class and a, b are individuals.

Definition 5.1.18 ($\mathcal{KB}_2^{DLP}$) *A DLP knowledge base $\mathcal{KB}_2^{DLP}$ is a pair $\langle \mathcal{T}_2^{DLP}, \mathcal{A}_2^{DLP} \rangle$.*

5.1.4. Prolog ($\mathcal{L}_3$)

$\mathcal{L}_3$ presents a DL language which allows the construction of class equivalence axioms that can be translated into Prolog(=,IC) programs. The extensions $\mathcal{L}_3^L$ and $\mathcal{L}_3^R$ allow to formulate class descriptions that can appear on the left ($\mathcal{L}_3^L$) or right ($\mathcal{L}_3^R$) hand side of DL class inclusion axioms.

Definition 5.1.19 ($\mathcal{L}_3$) *Class descriptions in $\mathcal{L}_3$ are formed according to the syntax rules of $\mathcal{L}_2$ and the additional rule:*

C, D	$\rightarrow$	$\exists R.C$	|	*(existential restriction)*
		$\geq 1\, R$	|	*(Min. Cardinality 1)*

Definition 5.1.20 ($\mathcal{L}_3^R$) *Class descriptions in $\mathcal{L}_3^R$ are formed according to the syntax rules of $\mathcal{L}_2^R$ and $\mathcal{L}_3$ and the additional rule:*

C, D	$\rightarrow$	$\geq n\, R$	|	*(Min. Cardinality n)*

Definition 5.1.21 ($\mathcal{L}_3^L$) *Class descriptions in $\mathcal{L}_3^L$ are formed according to the syntax rules of $\mathcal{L}_2^L$.*

Definition 5.1.22 ($\mathcal{T}_3^{DLP}$) *A DLP TBox $\mathcal{T}_3^{DLP}$ is a set of DLP TBox axioms (cf. Table 5.1 on page 116) where C is a $\mathcal{L}_3^L$ class, D is a $\mathcal{L}_3^R$ class, E, F are $\mathcal{L}_3$ classes and P, Q are properties.*

Definition 5.1.23 ($\mathcal{A}_3^{DLP}$) *A DLP ABox $\mathcal{A}_3^{DLP}$ is a set of $\mathcal{A}_2^{DLP}$ axioms.*

Definition 5.1.24 ($\mathcal{KB}_3^{DLP}$) *A DLP knowledge base $\mathcal{KB}_3^{DLP}$ is a pair $\langle \mathcal{T}_3^{DLP}, \mathcal{A}_3^{DLP} \rangle$.*

5.1.5. Exchanging DLP Ontologies

In order to facilitate the specification of DLP ontologies, we adapt the abstract syntax of OWL (Patel-Schneider et al., 2003) for the purposes of DLP. The necessary adaptations to make the abstract syntax concrete have been presented in (Bechhofer et al., 2003a), which is restricted to permissible $\mathcal{L}_3$ elements in Appendix A on page 233. The exchange of DLP ontologies using an RDF-based syntax can then make direct use of the abstract syntax to RDF statement mapping presented in (Patel-Schneider et al., 2003).

5.2. Semantics

We have several alternatives to assign semantics to the class descriptions of a DLP language $\mathcal{L}$. Firstly, we can assign semantics directly using a Tarski-style model theory such as done in Section 2.4.2 on page 34. Secondly, we can provide a reduction to some other logic such as done in Section 3.4.3 on page 64 for OWL, since we defined syntactic subsets of $\mathcal{SHIN}\mathcal{O}(\mathbf{D})$. Thirdly, we can rely on the translation function ϕ presented in Section 4.4 on page 87 and translate into FOL formulae, which in turn relies on the semantics of FOL. By construction, all three alternatives clearly yield the same semantics.

A fourth alternative is more convenient for our purpose, namely to rely on a translation function $\phi_{\mathcal{LP}}$ which directly translates into LP programs.

5.2.1. Translation of Classes

We adapt the approach of (Borgida, 1996) and define a recursive translation function $\phi_{\mathcal{LP}}$ which translates DL axioms of the form $C \sqsubseteq D$ and $C \equiv D$ into extended LP rules, which can be transformed into a conjunction (set of) LP rules using the well-known Lloyd-Topor transformations (Lloyd, 1987).

This translation function differs from the translation function of (Borgida, 1996) presented in Table 4.3 on page 90 by making the translation of constructors additionally dependent on the location of the constructor. Therefore, $\phi^L_{\mathcal{LP}}(C, x)$ ($\phi^R_{\mathcal{LP}}(C, x)$) uses the variable x to translate class descriptions C occurring on the left (right) hand side of an inclusion axiom into an LP formula.

Table 5.2 on page 120 describes the translation function $\phi_{\mathcal{LP}}$ and sorts the individual translation components according to the $\mathcal{LP}$ variant in which the resulting formula can be expressed. For the purpose of translation, we do not treat the core $\mathcal{L}_1$ languages separately, but duplicate the respective constructors into $\mathcal{L}^L_1$ and $\mathcal{L}^R_1$.

$$
\begin{array}{rcl}
\mathcal{LP}_0 = \text{Datalog} & & \\
\phi_{\mathcal{LP}}(C \equiv D) & \longrightarrow & \left\{ \begin{array}{l} \phi_{\mathcal{LP}}(C \sqsubseteq D) \\ \phi_{\mathcal{LP}}(D \sqsubseteq C) \end{array} \right. \\
\phi_{\mathcal{LP}}(C \sqsubseteq D) & \longrightarrow & \phi^L_{\mathcal{LP}}(C, y_i) \rightarrow \phi^R_{\mathcal{LP}}(D, y_i) \\
\phi^R_{\mathcal{LP}}(A, x) & \longrightarrow & A(x) \\
\phi^R_{\mathcal{LP}}((C \sqcap D), x) & \longrightarrow & \phi^R_{\mathcal{LP}}(C, x) \wedge \phi^R_{\mathcal{LP}}(D, x) \\
\phi^R_{\mathcal{LP}}((\forall R.C), x) & \longrightarrow & R(x, y_i) \rightarrow \phi^R_{\mathcal{LP}}(C, y_i) \\
\phi^R_{\mathcal{LP}}((\exists R.\{o\}), x) & \longrightarrow & R(x, o) \\
\phi^L_{\mathcal{LP}}(A, x) & \longrightarrow & A(x) \\
\phi^L_{\mathcal{LP}}((C \sqcap D), x) & \longrightarrow & \phi^L_{\mathcal{LP}}(C, x) \wedge \phi^L_{\mathcal{LP}}(D, x) \\
\phi^L_{\mathcal{LP}}((C \sqcup D), x) & \longrightarrow & \phi^L_{\mathcal{LP}}(C, x) \vee \phi^L_{\mathcal{LP}}(D, x) \\
\phi^L_{\mathcal{LP}}((\exists R.C), x) & \longrightarrow & R(x, y_i) \wedge \phi^L_{\mathcal{LP}}(C, y_i) \\
\phi^L_{\mathcal{LP}}((\exists R.\{o\}), x) & \longrightarrow & R(x, o) \\
 & & \\
\mathcal{LP}_1 = \text{Datalog(=)} & & \\
\phi^L_{\mathcal{LP}}(\top, x) & \longrightarrow & x = x \\
\phi^L_{\mathcal{LP}}(\{o_1, \ldots, o_n\}, x) & \longrightarrow & x = o_1 \vee \ldots \vee x = o_n \\
\phi^R_{\mathcal{LP}}(\{o\}, x) & \longrightarrow & x = o) \\
\phi^R_{\mathcal{LP}}(\top, x) & \longrightarrow & x = x \\
\phi^R_{\mathcal{LP}}(\leq 1\, R, x) & \longrightarrow & (R(x, y_1) \wedge R(x, y_2)) \rightarrow (y_1 = y_2) \\
 & & \\
\mathcal{LP}_2 = \text{Datalog(=,IC)} & & \\
\phi^R_{\mathcal{LP}}(\neg A, x) & \longrightarrow & \neg A(x) \\
\phi^R_{\mathcal{LP}}(\bot, x) & \longrightarrow & \neg(x = x) \\
 & & \\
\mathcal{LP}_3 = \text{Prolog(=,IC)} & & \\
\phi^R_{\mathcal{LP}}((\exists R.C), x) & \longrightarrow & R(x, f(x)) \wedge \phi^R_{\mathcal{LP}}(C, f(x)) \\
\phi^R_{\mathcal{LP}}((\geq n\, R), x) & \longrightarrow & (\bigwedge_{1 \leq i < j \leq n} (\neg C(x) \vee \neg(f_i(x) = f_j(x))) \wedge \\
 & & \bigwedge_{i \in [1,n]} (\neg C(x) \vee R(x, f_i(x))))
\end{array}
$$

Table 5.2.: Translation of Class Constructors to Logic Programs

The $\phi_{\mathcal{LP}}(C \equiv D)$ function decomposes class equivalence axioms into two inverse inclusion axioms. The later are translated by $\phi_{\mathcal{LP}}(C \sqsubseteq D)$, which calls (recursively) the respective $\phi^L_{\mathcal{LP}}(C, x)$ and $\phi^R_{\mathcal{LP}}(D, x)$ functions depending on the constructor, which is used in the description of C and D.

In the table we use our standard notation, viz. A is an atomic class name, C and D are class descriptions, R is a property and x, y_i are variables. It is important to note that y_i always refers to a fresh variable, which has not been previously used in the formula.

We further assume in the definition of $\phi_{\mathcal{LP}}$, that the preprocessing techniques described in Section 4.3 on page 83 have been applied beforehand. $\phi_{\mathcal{LP}}$ therefore does not include translations for constructors that are equivalent to other constructors and can be constructed from the combination. For example $\forall R.\bot$, which has been explicitly stated in Table 4.7 on page 111 is not explicitly mentioned as a translation component of Datalog(=,IC). The $\phi^R_{\mathcal{LP}}(\forall R.\bot, x)$ translation rather follows from the combination of $\phi^R_{\mathcal{LP}}(\forall R.C, x)$ and $\phi^R_{\mathcal{LP}}(\bot, x)$.

Example 5.2.1 *The following DL axiom*

$$A \sqcap \exists R.C \sqsubseteq B \sqcap \forall P.D$$

would be translated by $\phi_{\mathcal{LP}}$ into the extended LP formula

$$(A(x) \wedge R(x, y_1)) \wedge C(y_1) \rightarrow (P(x, y_2) \rightarrow (B(x) \wedge (D(y_2))))$$

We can now guarantee by the definition of $\phi_{\mathcal{LP}}$ and the Lloyd-Topor transformations (Lloyd, 1987) that this formula can be transformed into a conjunction of Horn formulae, viz. the following two of Datalog rules

$$B(x) \texttt{:-} A(x), R(x, y_1), C(y_1).$$
$$D(y_2) \texttt{:-} A(x), R(x, y_1), C(y_1), P(x, y_2).$$

5.2.2. Translation of Properties and Assertions

The translation of properties immediately follows from Table 4.6 on page 110, with the obvious restriction that the (inverse) functionality characteristic can only be expressed in Datalog(=) or more expressive LP languages.

For the translation of ABox assertions, we can immediately reuse the results of Section 4.5.1.2 on page 94. Obviously, individual equality assertions can only be expressed in Datalog(=) or more expressive LP languages. Individual inequality assertions are only possible in Datalog(=,IC) or more expressive LP languages.

Table 5.3 on page 122 describes the extension of the translation function $\phi_{\mathcal{LP}}$ for the translation of properties and assertions.

$$
\begin{array}{rcl}
\mathcal{LP}_0 = \text{Datalog} & & \\
\phi_{\mathcal{LP}}(P \equiv Q) & \longrightarrow & \left\{ \begin{array}{l} \phi_{\mathcal{LP}}(P \sqsubseteq Q) \\ \phi_{\mathcal{LP}}(Q \sqsubseteq P) \end{array} \right. \\
\phi_{\mathcal{LP}}(P \equiv Q^-) & \longrightarrow & \left\{ \begin{array}{l} Q(y,x) \rightarrow P(x,y) \\ P(y,x) \rightarrow Q(x,y) \end{array} \right. \\
\phi_{\mathcal{LP}}(P \sqsubseteq Q) & \longrightarrow & Q(x,y) \rightarrow P(x,y) \\
\phi_{\mathcal{LP}}(P^+ \sqsubseteq P) & \longrightarrow & P(x,y) \rightarrow P(x,z) \wedge P(z,y) \\
\phi_{\mathcal{LP}}(D(a)) & \longrightarrow & \phi^R_{\mathcal{LP}}(D,a) \\
\phi_{\mathcal{LP}}(P(a,b)) & \longrightarrow & P(a,b) \\
\mathcal{LP}_1 = \text{Datalog(=)} & & \\
\phi_{\mathcal{LP}}(a=b) & \longrightarrow & a=b \\
\mathcal{LP}_2 = \text{Datalog(=,IC)} & & \\
\phi_{\mathcal{LP}}(\neg(a=b)) & \longrightarrow & a \neq b
\end{array}
$$

Table 5.3.: Translation of Properties and Assertions to Logic Programs

5.2.3. Postprocessing

As we saw in Section 4.5 on page 93, extended LP formulae of the following basic structural types occur and have a well-defined transformation (Lloyd, 1987) into conjunctive normal forms:

$$
\begin{array}{lrcl}
(1) & B \rightarrow (H \wedge H') & \Leftrightarrow & (B \rightarrow H) \wedge (B \rightarrow H') \\
(2) & B \rightarrow (H' \rightarrow H) & \Leftrightarrow & (B \wedge H') \rightarrow H \\
(3) & (B \vee B') \rightarrow H & \Leftrightarrow & (B \rightarrow H) \wedge (B' \rightarrow H) \\
(4) & B \rightarrow \neg H & \Leftrightarrow & (B \wedge H) \rightarrow
\end{array}
$$

Since we apply the structural transformation as a preprocessing step (cf. Section 4.3 on page 83), we do not have to apply these transformations repeatedly, but can generate the target LP rules in one pass. We can additionally make use of these well-known transformations and will present a procedural variant of the translation function in Section 7.4.3 on page 183, which incorporates the a-posteriori transformation directly into the translation. This algorithmic version of the translation also incorporates the skolemization, which is needed to translate $\mathcal{L}_3$. The reader may wonder why we did not immediately present this integrated algorithm here.

The primary reason is that we can make use of $\phi_{\mathcal{LP}}$ for answering queries in Section 5.4 on page 130 and need a different transformation there, for which $\phi_{\mathcal{LP}}$ is more convenient and simpler to use. Additionally, as mentioned before, further optimizations of the LP program, e.g. the removal of unnecessary equivalences, can be achieved and can depart more effectively from extended LP programs.

5.2.4. Translation Semantics

We can now utilize $\phi_{\mathcal{LP}}$ to define the semantics of $\mathcal{KB}_i^{DLP}$ knowledge bases.

Definition 5.2.1 (Translation Semantics) *Given an interpretation $\mathfrak{I}$, $\mathfrak{I}$ satisfies $\mathcal{KB}_i^{DLP}$, iff.*

$$\mathfrak{I} \models \forall \bigwedge \{\phi_{\mathcal{LP}}(Norm(\phi))|\ \phi \in \mathcal{KB}_i^{DLP}\}$$

where the prefix $\forall$ is a sequence of universal quantifications for all variables in the conjunction and ϕ is an axiom in $\mathcal{KB}_i^{DLP}$.

Remark 5.2.1 *The reader may note that we consider the FOL interpretation of $\mathcal{KB}_i^{DLP}$ to be normative. If the semantics of a given logic database departs from the FOL semantics, the DLP interpretation provided by the system may not be correct.*

The following Lemma is a direct consequence of the construction of Norm(), $\mathcal{KB}_i^{DLP}$ and $\phi_{\mathcal{LP}}$ and Lloyd-Topor transformations (Lloyd, 1987):

Lemma 5.2.1 ($\mathcal{LP}$ Transformation) *$\forall \bigwedge \{\phi_{\mathcal{LP}}(Norm(\phi))|\ \phi \in \mathcal{KB}_i^{DLP}\}$ can be transformed into a $\mathcal{LP}_i$ rule set.*

5.2.5. Terminological Cycles

We have seen in Section 2.4 on page 31 that terminological cycles greatly increase the expressive power of Description Logics and their presence takes most logics into EXPTIME completeness. As we discussed in Section 2.4.2.1 on page 36, in case of cyclic TBoxes no unique models for a knowledge base necessarily exist. In this case, modern DL systems consider all possible interpretations by assigning descriptive semantics to TBoxes. The question therefore arises, whether the evaluation which we can expect in Logic Programming systems yields the same semantics.

Unfortunately, the *general* answer is no, since logic databases usually compute the Herbrand interpretation of some knowledge base by fixpoint iteration until the least fixpoint is reached.

Must the general answer, however, be given in our case ? Fortunately, the answer is no again for all DLP languages. This is due to the fact, that we have only very simple core languages available to state terminological cycles. Therefore, in these languages only one case of cycles studied in the literature (Nebel, 1991) appears. The following example demonstrates this case, which is called *component circular classes*.

Example 5.2.2 *In the following class equivalence axioms, circular definitions of* MAN *and* MALE *are made:*

$$\text{MAN} \equiv \text{HUMAN} \sqcap \text{MALE}$$
$$\text{MALE} \equiv \text{HUMAN} \sqcap \text{MAN}$$

$\phi_{\mathcal{LP}}$ *would translate these axioms into the following set of LP rules:*

$$\text{MAN}(x) \texttt{:-} \text{HUMAN}(x), \text{MALE}(x).$$
$$\text{MALE}(x) \texttt{:-} \text{HUMAN}(x), \text{MAN}(x).$$
$$\text{MALE}(x) \texttt{:-} \text{MAN}(x).$$
$$\text{MAN}(x) \texttt{:-} \text{MALE}(x).$$
$$\text{HUMAN}(x) \texttt{:-} \text{MAN}(x).$$
$$\text{HUMAN}(x) \texttt{:-} \text{MALE}(x).$$

We can easily see from the rule translation that MAN *and* MALE *are actually equivalent since their extensions will always be equivalent in the logic program.*

From the example, we can directly derive the syntactic criteria for component circularity, which are spelled out in the following definition:

Definition 5.2.2 (Component Usage + Circularity) *An atomic class* A_i directly uses *an atomic class* A_{i+1}*, if* A_i *is the left-hand side of a class equivalence axiom and* A_{i+1} *appears on the right-hand side of that axiom.* Uses *is the transitive close of the directly uses relation. A class* A *is* component-circular *iff* A *uses itself as a component.*

Under descriptive semantics, a set of component-circular classes behaves as if an unique, fresh atomic class is used in the definition of all classes in the component circle. Moreover, the following theorem shows that we can transform the TBox into such a form without changing "relevant parts" of the models, viz. turn the TBox into an acyclic, definitorial TBox by removing all component cycles.

Let $\mathbf{A}, \mathbf{R}$ be the set of classes and properties in a DLP TBox $\mathcal{T}$ and let $\mathfrak{I}|_X$ denote the restriction of an interpretation to a certain set $X \subseteq \mathbf{A} \cup \mathbf{R}$.

(Nebel, 1991)[pp.18-19] proofs the following theorem:

Theorem 5.2.1 (Component-cycle removal) *For every TBox* $\mathcal{T}$ *over* $\mathbf{A}$ *and* $\mathbf{R}$ *exists a TBox* $\mathcal{T}'$ *over* $\mathbf{A}'$ *and* $\mathbf{R}$*, where* $\mathbf{A} \subseteq \mathbf{A}'$*, such that for every admissible model* $\mathfrak{I}$ *of* $\mathcal{T}$ *there exists an admissible model* $\mathfrak{I}'$ *of* $\mathcal{T}'$ *and vice versa, with* $\mathfrak{I}|_{\mathbf{A}\cup\mathbf{R}} = \mathfrak{I}'|_{\mathbf{A}\cup\mathbf{R}}$ *and* $\mathcal{T}'$ *does not contain a component cycle.*

Theorem 5.2.1 shows that our intuition that was spelled out in Example 5.2.2 was correct. Obviously, the extensions of all atomic concepts in a set of mutually

component-circular classes have to be identical. Moreover, the extensions of these atomic classes have to obey the restrictions defined in the non-component-circular parts of all class definitions that participate in the component-circularity.

We can therefore summarize that component-circular classes do not add anything to the expressiveness since they can be eliminated without affecting the rest of the TBox. This elimination makes the TBox acyclic. In this case, however, the least-fixpoint semantics applied in logic databases agrees with the descriptive semantics, as both agree on the base interpretation taken for acyclic TBoxes (cf. Section 2.4.2.1 on page 36).

Restriction Circular Classes In $\mathcal{L}_3$ additionally another case of cycles can appear, which (Nebel, 1991) called *restriction circular classes*. A prototypical example for restriction circularity is the following definition $C \equiv \exists R.C$. Again, however, we do not really have a problem, since $\mathcal{L}_3$ is monotone, i.e. no negation can occur in circles. The reader may note that we do not use negation-as-failure in our programs, which are non-monotonic. Any occurrence of negation in the logic program is translated to integrity constraints, which are monotonic in their FOL translation. If an integrity constraint is violated, the whole program has no model, i.e. everything follows from the ontology.

In the case of monotonicity, the Knaster-Tarski theorem guarantees that every monotone fixpoint function has a least and greatest fixpoint. Following Lemma 2.3.1 on page 22 the descriptive semantics of OWL therefore agrees with the Herbrand model, which is obtained for the logic program through least fixpoint iteration (cf. Proposition 2.3.2 on 22), i.e. the entailment of ground facts under least fixpoint semantics matches the entailment under descriptive semantics.

5.3. Expressivity

This section compares the expressivity of the $\mathcal{L}_i$ languages with the different proposals for ontology languages, i.e. RDFS, OWL Lite and OWL DL. The asymmetry of the different $\mathcal{L}_i$ make these languages rather unusual by Description Logic standards, and the inability to represent universal property restrictions also leads to the situation that the languages have to depart from the usual Description Logic naming scheme. In consequence, we cannot simply compare the languages by naming the symbols for class constructors, which are provided by the language. Moreover, it is clear from the definition of the $\mathcal{L}_i$ languages that the languages are less expressive than any variant of OWL.

Therefore two main questions arise. Firstly, how expressive are the individual languages in comparison to other modeling languages we know, i.e. frame-based languages and RDFS? Secondly, how severe is the restricted expressivity of $\mathcal{L}_i$ in practise, i.e. how big is the percentage of currently available ontologies that can be expressed using $\mathcal{L}_i$?

5.3.1. Expressivity in Theory

5.3.1.1. $\mathcal{L}_0$

It is easy to see that even $\mathcal{L}_0$ includes RDF Schema,[1] since all RDFS classes are $\mathcal{L}_0$ classes and $\mathcal{T}_0^{DLP}$ includes all axioms needed to capture both class and property hierarchies and global domain and range restrictions such as available in RDFS.

$\mathcal{T}_0^{DLP}$ additionally captures the part of OWL DL that corresponds to a simple frame language, i.e. axioms that all define a primitive hierarchy of classes, where each class is defined as a frame. A frame specifies the set of subsuming classes and a set of slot constraints, which can be stated by an $\mathcal{T}_0^{DLP}$ axiom of the form $A \sqsubseteq \sqcap_i C_i$, where each C_i is a $\mathcal{L}_0^R$ class. In particular, frames can be associated with static attributes through the $\exists R.\{o\}$ constructor. Additionally, $\mathcal{T}_0^{DLP}$ allows to express most of the OWL DL property characteristics such as property inverse and symmetric and transitive properties. $\mathcal{L}_0^R$ further allows to state value restrictions to specify necessary conditions for class membership. $\mathcal{L}_o^L$ allows to state existential restrictions to specify sufficient conditions for class membership. Both restrictions can alternatively be understood as primitives that allow to state *local* domain and range constraints for properties (cf. Table 5.4)

Class Definition Axioms	DLP Translation
`Class` (A `localdomain`(P D) `localrange`(P R))	$\exists P.A \sqsubseteq D$ $A \sqsubseteq \forall P.R$

Table 5.4.: Additional DLP Convenience Primitives

The local range constraint is especially useful for a conversion of legacy object data models (Volz et al., 2002b) such as the ODMG object database models (Cattell et al., 2000) or the UML meta-modeling language (Object Management Group, 2003) for

[1] Under the practical assumption that all constraints stated in the definition of OWL DL (cf. Definition 3.4.4 on page 62) are applied with the appropriate adaptations to the RDFS vocabulary. This assumption is met by most if not all currently available RDFS vocabularies.

bootstrapping Semantic Web ontologies. Table 5.5 shows the translation of an Object Definition Language (ODL) class specification[2] to DLP axioms.

ODMG ODL Definition:

```
class Composer extends Person
( extent musicians)
{
    relationship set<Composition> composes
         inverse Composition::isComposedBy;
    attribute Instrument plays;
}
```

DLP Axioms :

```
class(Composer partial Person
    localrange(composes Composition)
    localrange(plays Instrument)
)
ObjectProperty(composes inverseOf(isComposedBy))
```

Table 5.5.: Translation of ODMG ODL Class Specification to DLP Axioms

5.3.1.2. $\mathcal{L}_1$

The main contribution of $\mathcal{L}_1$ is equality of individuals on the ABox side and the singleton set constructor, which allows to introduce nominals into the terminology on the TBox side. $\mathcal{L}_1^R$ allows to specify maximum cardinality restrictions with value 1. Therefore all OWL Lite property characteristics can be represented. Functional properties allow uniqueness conditions to be captured, which is particularly useful to entail the equality of objects that use the same objects as values of a property. In a sense, they thereby capture the uniqueness aspect of primary keys in relational databases.

$\mathcal{A}_1^{DLP}$ additionally allows to instantiate individual equivalence axioms. The evaluation in chapter 8 will show that this small increase of expressivity comes at the

[2] The `extends` keyword allows to specify subsumption in ODL. The reader may note that the `extent` keyword has a different role and specifies the physical table, in which all objects of a class should be stored.

high price of having to handle the equivalence theory, which severely affects performance.

5.3.1.3. $\mathcal{L}_2$

$\mathcal{L}_2^R$ allows to define basic constraints on classes, and allows to make a given knowledge base inconsistent. It is the first language where classes are not necessarily satisfiable due to the constraints that have been formulated in inclusion axioms.

5.3.1.4. $\mathcal{L}_3$

$\mathcal{L}_3$ greatly increases the expressive power by allowing existential property restrictions on the right-hand side of inclusion axioms. $\mathcal{L}_3$ thereby allows to state necessary conditions on the existence of objects. This enables the modeling of incomplete information, allowing a distinction to be made between objects for whom all relations to other objects are known and objects that are only incompletely specified.

5.3.1.5. DLP vs. OWL

The asymmetry of the DLP language primitives, viz. the fact that most DL class constructors can only be used on the right-hand side of inclusion axioms, makes DLP formally a much less expressive ontology language than OWL Lite, which corresponds to $\mathcal{SHIF}(\mathbf{D})$ and theoretically (but not syntactically) allows to use all DLP language constructors (except for $\geq n\, R$) on both sides of inclusion axioms and in class equivalence axioms. The next section will show, however, that most OWL ontologies only use the language primitives provided by DLP. This is mainly due to the fact that most OWL classes are primitive classes and defined through inclusion axioms, i.e. the expressiveness of the right-hand side of inclusion axioms is central in practical usage, while atomic class names are the most frequently used class constructor on the left-hand side of inclusion axioms.

5.3.2. Expressivity in Practise

In order to assess the expressivity of the $\mathcal{L}_i$ languages in practise, we have to consider currently available ontologies. We restrict our attention to the ontology library provided by the DAML project, which contained references to 281 ontologies in DAML+OIL, RDFS, DAML-ONT, OIL and OWL (44) format.

Interestingly, many of the ontologies in the library turn out to be just conversions of ontologies, which were initially created in some other (mostly less expressive)

DLP Variant	Completely	μ # Axioms	σ # Axioms	Min. # Axioms
$\mathcal{L}_0$	77%	93%	0,0239	25%
$\mathcal{L}_1$	79%	95%	0,0151	25%
$\mathcal{L}_2$	82%	97%	0,0076	50%
$\mathcal{L}_3$	87%	99%	0,0011	75%

Table 5.6.: DAML.ORG Ontologies in DLP

representation language. For example, the famous wine ontology is with us since 'Classic' times (for more than 15 years !). Viz. the 'bootstrapping' motivation given in Section 1.2 on page 7 appears to be justified.

185 ontologies are specified in DAML+OIL and we attempted to convert them into OWL with the OilEd editor. Unfortunately, only 82 ontologies could be correctly translated into OWL. Due to some other errors (the most prominent being that ontologies could not be downloaded at the specified location (34) or did not validate as OWL DL (23)), only 112 ontologies could be evaluated in the end.

Our evaluation considers the percentage of all axioms in individual ontologies that can be expressed using $\mathcal{L}_i$, where class equivalence axioms $C \equiv D$ are converted to two inverse inclusion axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.

Table 5.6 summarizes the assessment of the OWL ontology library. The individual results can be found in appendix C on page 245. The most important result of the evaluation is that the expressivity of $\mathcal{L}_0$ suffices to express 77% of all ontologies in the library completely. While $\mathcal{L}_1$, which is able to express functional properties, provides little improvements to this number. Since $\mathcal{L}_2$ is able to catch OWL DisjointClass axioms, an additional number of ontologies can be expressed. $\mathcal{L}_3$, which can express incomplete information using existential restrictions in the head, catches another 5% of the assessed ontologies.

If we consider the expressivity of DLP from the perspective of number of axioms that can be expressed, DLP languages are able to capture the large majority of all axioms in an ontology, where $\mathcal{L}_0$ misses in average 7% of all expressed information, whereas $\mathcal{L}_3$ only misses 1% of the expressed axioms. The reader may note that the standard deviation (σ) is very low, i.e. the individual ontologies typically do not diverge very much from the average case. The minimum number of axioms, viz. the maximum deviation observed, can be accounted to one single ontology[3], which contained 16 inclusion axioms, 4 of which were only expressible in OWL DL. The results of the evaluation are very promising and show that DLP is fitted well for the Web ontologies available today. This shows that DLP is useful in practise and should be sufficient for most Semantic Web applications.

[3] `http://orlando.drc.com/daml/Ontology/TaskListUHTLScenario/current/`.

5.4. Reasoning Problems

As discussed in the introduction of Chapter 4 on page 79, one of the primary goals of this part is to enable some fragment of DL inferencing to be performed by LP engines. In this section we will discuss how the kinds of inferences which are typically of interest in DL, can be translated to query answering in LP. Although our emphasis will naturally reside on performing DL inferencing, via our translation $\phi_{\mathcal{LP}}$, using a LP system, the reverse mapping could be used to provide a restricted (Horrocks & Tessaris, 2002) form of LP inferencing using a DL reasoning engine.

On the one hand, DL reasoners typically support the various reasoning tasks that are described in Section 2.4.3 on page 38. We call $\mathcal{QDL}$ the query language which allows to invoke these reasoning tasks. On the other hand, LP engines typically support the two queries described in Section 2.3.4 on page 27. We call $\mathcal{QLP}$ the query language defined by these two kinds of LP queries.

In the following we will use the established notation. P and Q refer to properties, a and b denote individuals and C and D refer to $\mathcal{L}_i^{L|R}$ class definitions.

5.4.1. Reducing DL Queries to LP Queries

We will now discuss how we can reduce $\mathcal{QDL}$ queries stated on $\mathcal{KB}_i^{DLP}$ knowledge bases to $\mathcal{QLP}$ queries on the LP_i programs $\mathcal{LP}$, which are created by applying $\mathcal{LP} = \phi_{\mathcal{LP}}(\text{Norm}(\mathcal{KB}_i^{DLP}))$ and the necessary a-posteriori transformations (cf. Section 5.2.3 on page 122).

5.4.2. Class-related ABox Problems

We can easily see that we can only ask class-individual membership queries using class definitions available in $\mathcal{L}_i^L$. We can recall that $\mathcal{L}_i^L$ are classes which may appear on the left-hand side of class inclusion axioms and can therefore be translated in the body of LP rules. Therefore, they can appear in the open $\mathcal{QLP}$ atom queries, which are, as mentioned above, essentially goal clauses. This has serious consequences with respect to the expressiveness of the query language. For example, we can not ask queries for membership in the following classes:

- *negated classes*, e.g. who is not a woman?
- *value restricted classes*, e.g. who is an offspring of a musician dynasty ($\forall hasParent^+.Musician$) ?

- *max. cardinality restricted classes*, e.g. who plays at most one thing ($\leq 1\, plays$) ?

This might appear to be a serious restriction. However, in practise, this is not the case. Firstly, the answers to such queries in DL systems are often unintuitive,[4] since the open-world assumption is applied in Description Logics.

Example 5.4.1 (Open-world semantics) *Consider the following single ABox assertion*

HASCHILD(JOHANN-SEBASTIAN, WILHELM-FRIEDEMANN)

from Figure 2.5 on page 35. In a database and Logic Programming, where the closed-world assumption is applied, this assertion is a representation of the fact that JOHANN-SEBASTIAN *has indeed only one (known) child, namely* WILHELM-FRIEDEMANN. *In DL, however, it only asserts that* WILHELM-FRIEDEMANN *is* a *child of* JOHANN-SEBASTIAN.

Hence, JOHANN-SEBASTIAN *could always have more children.*[5] *If we would ask for those individuals in the ABox who have at most one child,* JOHANN-SEBASTIAN *would not be among the given answers, since the assertion* ≤ 1 HASCHILD(JOHANN-SEBASTIAN) *has not been made in the ABox.*

Secondly, the ABox querying capabilities of Description Logics are very limited when compared to Turing complete languages like SQL and Prolog. All $\mathcal{QDL}$ instance retrieval problems, given that the class C whose extension is to be retrieved is an $\mathcal{L}_i^L$-class, can nevertheless be reduced to $\mathcal{QLP}$ queries.

If C is a complex $\mathcal{L}_i^L$ class description, we will reduce the $\mathcal{QDL}$ query into several $\mathcal{QLP}$ queries. The reader may note that we could similarly invent a new atomic class name Q, use C and Q in a new DL inclusion axiom $C \sqsubseteq Q$ and ask one single $\mathcal{QDL}$ query for the atomic class Q.

However, we do not make this simplistic translation, since the translation involves a manipulation of the TBox and, consequently, a manipulation of the rules in $\mathcal{LP}$. The behavior of LP systems in this case is not clear. Some systems simply disallow a (programmatic) change in the rules of (running) logic programs. Others allow such a change but require a recompilation of the program (e.g. CORAL (Ramakrishnan et al., 1994)), which is clearly more expensive than asking several $\mathcal{QLP}$ queries.

Remark 5.4.1 *The reader may also note that our reduction of* $\sqcup$ *is only complete for the DLP fragments, since it is impossible to instantiate disjunctions in individual assertions and* $\sqcup$ *is not available on the right-hand side of inclusion axioms.*

[4] At least to non-logicians.

[5] In fact, Bach had **twenty** children, seven with his first wife and cousin Maria Barbara and thirteen children with his second wife Anna Magdalena.

5.4.2.1. Instance checking

$\mathcal{QDL}$ instance checking refers to the problem of determining whether an individual a is an instance of C. We translate this query to $\mathcal{QLP}$ ground atom queries in a recursive fashion. If:

- C *is a class name*: The $\mathcal{QDL}$ query $C(a)$ is directly translated into the $\mathcal{QLP}$ ground atom query $:\text{-}C(a)$. The $\mathcal{QDL}$ query is answered positively (negatively), if the $\mathcal{QLP}$ query has a positive (negative) answer.
- $C = C_1 \sqcap \ldots \sqcap C_n$: The initial $\mathcal{QDL}$ query is translated to n $\mathcal{QDL}$ queries $C_i(a)$, which must all be answered positively to obtain a positive answer on the initial query. Otherwise, a negative answer is given for the initial query.
- $C = C_1 \sqcup \ldots \sqcup C_n$: The initial $\mathcal{QDL}$ query is translated to n $\mathcal{QDL}$ queries $C_i(a)$. At least one of these $\mathcal{QDL}$ queries must be answered positively to obtain a positive answer for the initial $\mathcal{QDL}$ query. We can process the multiple $\mathcal{QDL}$ queries sequentially and stop processing these queries as soon as we found one positive answer.
- $C = \{o_1, \ldots, o_n\}$: The initial $\mathcal{QDL}$ query is directly translated into multiple $\mathcal{QLP}$ ground queries

  $$:\text{-}o_1 = a.$$
  $$\ldots$$
  $$:\text{-}o_n = a.$$

  At least one of these $\mathcal{QLP}$ queries must be answered positively to obtain a positive answer for the initial $\mathcal{QDL}$ query. We can process the multiple $\mathcal{QLP}$ queries sequentially and stop processing these queries as soon as we found one positive answer.
- $C = \exists R.D$: The initial *ground* $\mathcal{QDL}$ query is translated into an *open* $\mathcal{QLP}$ query

  $$:\text{-}R(a, y_1), D(y_1).$$

  We can give a positive answer if we find substitutions obtained for the variable y_1.

5.4.2.2. Instance Retrieval

The $\mathcal{QDL}$ instance retrieval problem asks for all individuals that are members of a class C. This query is reduced to multiple open $\mathcal{QLP}$ atom queries. This reduction is performed using the $\phi_{\mathcal{LP}}$ translation of C and an answer variable x which will be

used as a container for all individuals that constitute the answer of the $\mathcal{QDL}$ query. We obtain a FOL formula

$$\phi = \phi^L_{\mathcal{LP}}(C, x)$$

and translate ϕ into disjunctive normal form (DNF)

$$\phi \equiv \bigvee_i K_i$$

Each disjunct K_i then forms a $\mathcal{QLP}$ query, by considering K_i as the body of a LP goal clause

$$\texttt{:-}\, K_i.$$

The answer of the initial $\mathcal{QDL}$ query is the union of all substitutions obtained for the variable x in each individual $\mathcal{QDL}$ query K_i.

The construction of the DNF can be achieved using a repeated (recursive) application of one of the laws of distribution:

$$F \vee (G \wedge H) \Rightarrow (F \wedge G) \vee (F \wedge H)$$

The simplicity of this solution can be accounted to the fact that all ϕ are simple formulae with the following two properties. Firstly, ϕ is a quantifier free formula, due to the fact that all terms produced by $\phi^L_{\mathcal{LP}}$ are expected to be part of the body of a LP rule[6]. Secondly, the $\phi^L_{\mathcal{LP}}$ translation rules, which are the only relevant rules for a $\mathcal{L}^L_i$ expression like C, only produce (nested) conjunctions and disjunctions, i.e. we only have to deal with positive literals.

These two characteristics allow us to use a "propositional" algorithm for the construction of a DNF form, i.e. all literals in ϕ are treated like predicates in propositional logic and restrict our attention to recursive application of the distribution law mentioned above.

Eventually we "multiply" $\vee$ out as often as necessary, i.e. push all $\wedge$ inwards. Obviously this multiplication is linearly bounded in the number of $\wedge$ occurring in the formula.

Example 5.4.2 *Consider the following open $\mathcal{QDL}$ class-individual membership query:*

$$Q = (\exists R.A \sqcap (B \sqcup C)) \tag{5.1}$$

The translation into a FOL formula provided by $\phi^L_{\mathcal{LP}}(Q, x)$ results in

$$R(x, y) \wedge (A(y) \wedge (B(y) \vee C(y))) \tag{5.2}$$

[6]They are thereby implicitly quantified.

which can be transformed in two steps into DNF

$$R(x,y) \wedge ((A(y) \wedge B(y)) \vee (A(y) \wedge C(y))) \tag{5.3}$$
$$(R(x,y) \wedge A(y) \wedge B(y)) \vee (R(x,y) \wedge A(y) \wedge C(y)) \tag{5.4}$$

We then issue two $\mathcal{QLP}$ queries

$$\texttt{:-}\, R(x,y), A(y), B(y). \tag{5.5}$$
$$\texttt{:-}\, R(x,y), A(y), C(y). \tag{5.6}$$

The answer is then given by union of the substitutions for the variable x in the answer of the two queries.

We could additionally make use of the laws of absorption to simplify the formula further and avoid redundant conjuncts. However, in practise, only few redundancies appear, since queries are usually not very complex terms.

5.4.2.3. Dual retrieval problem

The dual $\mathcal{QDL}$ retrieval problem asks for all classes that a given individual a instantiates. Apparently this amounts to issuing $n \times m$ $\mathcal{QDL}$ instance checking problems, where n is the number of classes in $\mathcal{T}_i^{DLP}$ and m the number of individuals in $\mathcal{A}_i^{DLP}$.

Obviously, we can do better than that, taking into consideration equivalence classes of individuals, where it is sufficient to test one representative and taking the class hierarchy into consideration. If we know, for example, that $C \sqsubseteq D$ and can show that a is a member of C, we can automatically deduce that a must also be a member of D.

These optimizations are similar to the optimizations taken for classification, which are described in more detail in the subsequent Section 5.4.4.3 on page 136.

5.4.3. Property-related ABox Problems

5.4.3.1. Ground property fillers

The case of determining ground property fillers is trivial as all properties are atomic, i.e. the ground $\mathcal{QDL}$ query $P(a,b)$ has a positive (negative) answer iff the $\mathcal{QLP}$ query

$$\texttt{:-}\, P(a,b).$$

has a positive (negative) answer.

5.4.3.2. Open property fillers

The case of open property fillers is similar to ground property fillers. The $\mathcal{QDL}$ query for all pairs (a, b) that are members of P is effected by translation into the open $\mathcal{QLP}$ query

$$:\text{-}\, P(x, y).$$

and returning the pairs of substitutions obtained for the variables x and y in the $\mathcal{QDL}$ query.

5.4.4. TBox Problems

The main reasoning problem for DLP, which does not feature full negation, is class subsumption. For $\mathcal{L}_i$-classes, $\mathcal{QDL}$ class subsumption queries can be reduced to $\mathcal{QLP}$ only by assertions. This is due to the fact that Logic Programs, unlike Description Logics, typically cannot reason over the conceptualization, viz. the rules of the program, itself. They can only manipulate the extensions of given predicates via the deduction process that is controlled by the evaluation of the program.

We can however add new assertions to the extension of some predicate and observe what happens with the new assertion, i.e. whether it "suddenly" appears in the extension of some other predicate. If this is the case, the extension of the other predicate must necessarily be a superset of the predicate to which we made the assertion initially. This conclusion only holds, since we do not have negation-as-failure in the rules generated by $\phi_{\mathcal{LP}}$, i.e. we have only "positive" flows of elements in the Herbrand universe.

5.4.4.1. Subsumption between two classes

We make use of the aforementioned observation. To test whether C is a subclass of D, we construct a prototypical instance a of C and check whether $\mathcal{KB}_i^{DLP} \cup \{C(a)\}$ entails $D(a)$. The reader may note that a must be a new constant that did not appear in $\mathcal{KB}_i^{DLP}$ before. After the check, $C(a)$ has to be removed from $\mathcal{KB}_i^{DLP}$.

The assertion approach has the obvious consequence that, in the case of complex class definitions, we can only detect whether a $\mathcal{L}_i^L$ class subsumes a $\mathcal{L}_i^R$ class. This is due to the fact that we can only formulate assertions using $\mathcal{L}_i^R$ classes and can only ask class-membership queries for $\mathcal{L}_i^L$ classes.

If C or D is a (complex) class description, we have to compile new rules. As mentioned before, the resulting change of the rule set in the logic program might make this operation very expensive in logic databases. We can therefore conjecture, that DL systems will outperform LP systems with respect to the subsumption problem.

5.4.4.2. Equivalence of two classes

To test whether $C \equiv D$, we have to check whether $\mathcal{KB}_i^{DLP} \cup \{C(a), D(b)\}$ entails $D(a)$ and $C(b)$. The reader may note that a and b must both be new constants that did not appear in $\mathcal{KB}_i^{DLP}$ before. After the check, the assertions $C(a)$ and $D(b)$ have to be removed from $\mathcal{KB}_i^{DLP}$.

5.4.4.3. Classification

Complete information about the class hierarchy can be obtained by computing the partial ordering of classes in $\mathcal{T}_i^{DLP}$ based on the subsumption relationship. This complete information is also required to answer queries about the position of a given class C in the hierarchy, viz. queries for all or only the most-specific (named) superclasses of C in $\mathcal{T}_i^{DLP}$ and for all or only the most-general (named) subclasses of C in $\mathcal{T}_i^{DLP}$.

Obviously, the classification of n classes in a TBox is at most $O(n^2)$, i.e. of quadratic complexity in the number of classes, since it requires $n \times (n-1)$ subsumption checks. We can, however, minimize this number further by taking the structure of classes into account and controlling the order in which classes are added to the hierarchy.

For our purpose we can reuse a plethora of optimizations (e.g. (Baader et al., 1994; Levinson, 1992; Lipkis, 1982; MacGregor, 1988; MacGregor, 1991)) devised in the implementations of Description Logic reasoners. Most optimizations are based on traversal of the class hierarchy and minimize the number of tests that are required in order to add a new class.

For example, (Lipkis, 1982) describes an optimization for classification in the very first KL-ONE implementation. His optimization is essentially a simple traversal method whose idea is to compute the superclasses of a class by searching down the hierarchy from the top node and the subclasses of a class by searching up the hierarchy from the bottom node.

Top search The top search takes advantage of the transitivity of the subsumption relation when a class A is classified by propagating failed classification results down the hierarchy. Hence, it deduces while avoiding the subsumption test itself, that if A is not subsumed by B, then it cannot be subsumed by any other class that is subsumed by B.

Bottom search Bottom search uses a similar technique and only tests whether B is a subclass of A if A is already known to subsume all classes that are subclasses of B. The bottom search is also limited by information from the top search.

Told subsumption Other enhancements to the classification can be made by "avoiding subsumption tests by exploiting relations which are obvious when looking at the syntactic structure" (Baader et al., 1994)[sec. 5 p. 24] For example, if the KB contains the inclusion axiom $A \sqsubseteq C$, then C is said to be a *told* superclass of A. Similarly, if C is a conjunctive class, all conjuncts of C will be superclasses of A. We can conjecture that told subsumption will be very efficient for the DLP languages, since our translation only yields inclusion axioms.

Apparently, it is easy to compile the list of told superclasses while reading in class definitions and storing this information in auxiliary, built-in system predicates for later usage. Eventually, we also need such auxiliary predicates to manage markers for the traversal method of (Lipkis, 1982) and can use told information to preset these markers.

Effect of optimizations The evaluation of (Baader et al., 1994) shows that the described optimizations safe an average of 80 % in the number of subsumption calls on the evaluated knowledge bases. In fact, their implementation of the optimizations were able to speed up the classification task of the KRIS system by an order of 3 magnitudes. We therefore consider the point for optimization taken and do not evaluate classification further in Chapter 8 on page 193.

5.4.4.4. Class Satisfiability Queries

As mentioned before, only $\mathcal{L}_i^R$ with $i \geq 2$ is expressive enough to make a given set of classes unsatisfiable. In order to check whether a class is satisfiable, we have to use assertions again.

Class satisfiability We can simply rely on the fact that the integrity constraints defined in the LP engine will disallow us to assert any information to an unsatisfiable class. If we try to assert a new individual a to be a member of class C and the system responds with an integrity constraint violation, we obviously detected that C can not be satisfied.

If C is a complex class definition, we have to compile new rules for C. Since we try to assert a new instance, C must be a $\mathcal{L}_i^R$ class. After the satisfiability check, we have to revoke the compiled rules and the made assertion (if it succeeded).

Apparently, satisfiability checking will be a very expensive operation in LP systems, since the change of the program typically results in recompilation of the program and optimizations based on the structure of the program are usually carried out.

Knowledge base satisfiability We can check whether the whole TBox $\mathcal{T}_i^{DLP}$ is satisfiable by asking whether $\top$ is satisfiable. Apparently, this amounts to an individual equivalence assertion $a = a$ for some new individual a that has not previously been an element in $\mathcal{KB}_i^{DLP}$.

Disjointness of classes The disjointness of classes can be detected by an attempt to make two assertions to classes using the same individual. If only one of the two assertion succeeds, we can deduce that the classes are disjoint. If both assertions succeed, the classes are not disjoint in all possible knowledge bases. If no assertion succeeds, both classes are unsatisfiable, i.e. equivalent to each other and $\bot$.

The reader may note that disjointness in DL does not denote that the extension of classes are disjoint in the particular knowledge base at hand. In Logic Programming this "other" form of disjointness can easily be computed, i.e. we can not find variable substitutions for x in the following rule $:\text{-}C(x), D(x)$..

5.4.4.5. Property Subsumption and Equivalence

Similarly to our approach to class subsumption, we could detect property subsumption by assertions.

In $\mathcal{KB}_0^{DLP}$ knowledge bases we all properties are atomic and property subsumption is only an asserted partial order on properties, we could detect property subsumption from the asserted definitions under the condition that we store the definition itself. This is possible since we can neither make the knowledge base unsatisfiable nor have interactions of equality with the property hierarchy, e.g. via functional properties.

Meta-Reasoning We can store the assertions about property subsumption using a dedicated predicate $\leq$ and axiomatize this predicate $\leq$ to capture the axioms of partial orders, viz. reflexivity, antisymmetry and transitivity. For convenience, we write the predicate infix as previously done for equivalence.

$$(x \leq x) :\text{-}. \quad reflexivity \tag{5.7}$$

$$(x = y) :\text{-} (x \leq y), (y \leq x). \quad antisymmetry \tag{5.8}$$

$$(x \leq y) \texttt{:-} (x = y). \tag{5.9}$$
$$(y \leq x) \texttt{:-} (x = y). \tag{5.10}$$
$$(x \leq z) \texttt{:-} (x \leq y), (y \leq z). \quad transitivity \tag{5.11}$$

A simulation of the $\mathcal{QDL}$ query for subsumption between the properties P and Q can then use a ground $\mathcal{QLP}$ query $\texttt{:-}(p \leq q)$.. The reader may note that $\leq$ is an auxiliary system predicate which is only used for answering $\mathcal{QDL}$ property subsumption and equivalence queries. $\leq$ is not a binary predicate that is defined through $\mathcal{T}_i^{DLP}$, rather the extension of $\leq$ is defined through $\mathcal{T}_i^{DLP}$.

Property Equivalence Likewise to class equivalence, the $\mathcal{QDL}$ query for the equivalence of properties can be reduced to two inverse $\mathcal{QDL}$ queries on the subsumption of the two properties.

5.4.4.6. Property Characteristics

We can ask more structural queries about properties, which are not typical DL reasoning problems, but in style of property subsumption. Again, we could think of two approaches which are based on either testing on new, hypothetical individuals or on predicates storing meta-data. We will see, however, that the meta-approach does not work for all cases, e.g. domains and ranges.

Domain and Range We can easily retrieve the set of asserted domain and range classes of a property using the meta-approach. However, due to the fact that these classes might have subclasses, we are not able to retrieve all possible (inherited) domains and ranges. To retrieve these classes, we need to have a complete classification of all classes available in the TBox and look up the classes which are subsumed by the asserted domains and ranges.

Alternatively, the assertion approach would proceed asserting a new pair (a, b) of new, previously unused individuals to be an instance of a property P and query all n $\mathcal{L}_i^L$ classes in the TBox whether a (b) appear in the extension of that class. If so, the respective class is the domain (range) of P. Apparently, we have to undo the property filler assertion $P(a, b)$ after the completion of the answer.

Inverse and Symmetric Querying properties for their asserted inverse properties naturally amends itself to the meta-data approach, since only the simple inference that inverse itself is symmetric has to be made to capture all inverses of a

property. The symmetry can easily be captured by the rule:

$$inverse(y, x) \text{ :- } inverse(x, y). \quad symmetry \tag{5.12}$$

$$inverse(x, z) \text{ :- } (x \leq y), inverse(y, z). \quad inheritance \tag{5.13}$$

A $\mathcal{QDL}$ query for the inverses of a property P would then be answered by returning the set of substitutions for the variable x in the open $\mathcal{QLP}$ query $\text{ :- } inverse(P, x)$.

Similarly, a $\mathcal{QDL}$ query for whether a given property P is symmetric would be answered by the closed $\mathcal{QLP}$ query $\text{ :- } inverse(P, P)$.

Transitivity Answering the $\mathcal{QDL}$ query for the transitivity of a property with the assertion approach requires three new, previously unused individuals which are asserted for two (connected) pairs of property fillers $P(a, b)$ and $P(b, c)$. We then have to issue a ground $\mathcal{QLP}$ atom query on whether $\mathcal{R} \cup \{P(a, b), P(b, c)\}$ entails $P(a, c)$. After the query, obviously, we have to revoke the assertions after answering the $\mathcal{QLP}$ query.

The meta-data approach requires the simple axiomatization that transitivity also holds for inverse properties.

$$isTransitive(x) \text{ :- } inverse(x, y), isTransitive(y). \tag{5.14}$$

5.4.5. Descriptive Information

As we have seen in the previous section, the representation of information that describes the structure of the knowledge base itself, can be used for some simple inferences. This meta-information, which we call *descriptive information*, is also necessary for the classification task where we need information about which classes and properties have been declared in the ontology.

Descriptive information is not only useful for system internal purposes such as characterized above, but also from a user perspective. Users are required to know the TBox in order to query the ABox in a sensible manner. Therefore they need to have access to information about the definitions of classes and properties. This would also allow to represent additional "non-logical" information similar to AnnotationProperties in OWL (McGuinness & van Harmelen, 2003), which can help users in understanding class and property definitions made in a knowledge base $\mathcal{KB}_i^{DLP}$.

We have several alternatives for the representation of descriptive information. For example, one can utilize the fact that the specifications in $\mathcal{KB}_i^{DLP}$ itself are made using a syntactic subset of the primitives available in OWL. We can therefore rely

on the RDF representation of OWL and simply store the statements of the RDF graph in a dedicated ternary predicate, e.g. $statement$. Alternatively, we can use the EBNF described in Appendix A on page 233 and derive a relational storage structure of the grammar. Similarly, we could directly use the EBNF and store the nested structure in LP systems that allow for nested data structures, i.e. Prolog and many deductive databases such as Coral (Ramakrishnan et al., 1994).

It is important to note that we only represent this descriptive information for tasks that do not require sophisticated logical reasoning but simple database-like lookup of information. We do not control the reasoning itself via an axiomatization of descriptive information such as done in (Zou, 2001; Roo, 2002; Fikes & McGuiness, 2001; Weithöner et al., 2003).

We can however devise other auxiliary predicates that allow to cache inferred information, e.g. deduced subsumption relationships, which hold as long as the TBox itself is not altered. Since updates on the TBox require a recompilation of the whole logic program anyways, the classification and subsumption information holds for the life time of the program. To summarize, descriptive information has the same role as descriptive information in normal databases.[7]

5.5. Complexity

To determine the complexity of the $\mathcal{L}_i$ languages, we can reuse the properties of the translation function $\phi_{\mathcal{LP}}$ and make use of the complexity results known for different variants of Logic Programming, which were mentioned in Section 2.3.5 on page 27.

5.5.1. Datalog-variants of DLP

5.5.1.1. ABox reasoning problems

Sections 5.4.2 and 5.4.3 have shown how a $\mathcal{QDL}$ ABox reasoning problem can be translated into multiple $\mathcal{QLP}$ queries. For answering $\mathcal{QLP}$ queries, we can recall from Section 2.3.5 on page 27 that (Vardi, 1982; Immerman, 1986) showed that the data complexity, i.e. the complexity of answering $\mathcal{QLP}$ queries of Datalog is P hard. We will make this bound more crisp for DLP. As a first step we characterize what *polynomial* means for arbitrary Logic Programs $\mathcal{LP}$. (Vardi, 1982)[Thm. 5, pp. 5-6] proves the following theorem:

[7] In Oracle, most databases employ a special reserved table, e.g. `syscat.tables`, to store information about user defined tables.

Theorem 5.5.1 (Vardi-Immerman) *The data complexity of any Datalog $\mathcal{LP}$ and input database D is bounded by*

$$O(n^{k+1})$$

where k is the maximum number of variables in rules of $\mathcal{LP}$ and n is the size of the extensional database D.

Corollary 5.5.1 *The data complexity of DLP knowledge bases $\mathcal{KB}^{DLP}_{i\leq 2}$ is then bounded by*

$$O(|\mathcal{A}^{DLP}_{i\leq 2}|)^4$$

Proof: Clearly, the size of the extensional database D is equivalent to $n = |\mathcal{A}^{DLP}_{i\leq 2}|$, i.e the number of property fillers, individual assertions and individual (in)equivalences, since we can assume, without loss of generality, that all assertions are made on atomic classes (cf. Section 4.5.1.2 on page 94) and atomic properties. The size of the input database is therefore equivalent to the size of the number of axioms in $\mathcal{KB}^{DLP}_{i\leq 2}$.

Every $\mathcal{LP}$ created by $\phi_{\mathcal{LP}}$ has at most 3 variables k in its rules. In fact, the bound $k = 3$ arises from the translation of transitive properties. Since we apply structural transformation (Plaisted & Greenbaum, 1986), i.e. assign a new name for each instantiation of a $\mathcal{L}_i$ constructor in a (complex) class description, we only have to look at the individual $\mathcal{L}_i$ constructors. The definition of $\phi_{\mathcal{LP}}$ then immediately shows that only the translation of $\exists R.C$,$\forall R.C$ and $\geq n\, R$ lead to $k = 2$ variables. The boolean constructors $\sqcap, \sqcup, \neg$ only require $k = 1$ variable. $\leq 1\, R$, however, requires $k = 3$ variables. Similarly, the transitivity of the equality theory requires $k = 3$ variables. For DLP, we can therefore deduce that we can rewrite every program into a form such that k equals 3. Substituting k and n into the upper bound of Theorem 5.5.1 then shows $O(|\mathcal{A}^{DLP}_{i\leq 2}|)^4$.

□

5.5.1.2. TBox Reasoning Problems

The complexity of solving TBox reasoning problems is harder to assess, since the answering of $\mathcal{QDL}$ queries requires manipulation of the logic program by asserting and retracting facts or even rules. In this case, program complexity is a more precise measure, which is EXPTIME in the case of Datalog. This result is additionally backed up by Bonner in (Bonner, 1992), who shows that Datalog augmented with an operator for modifying the logic program (called substitution) has EXPTIME complexity.

5.5.2. Prolog-variant of DLP

We can recall that $\mathcal{L}_3$ knowledge bases are compiled into Prolog programs. Prolog itself is undecidable, since it can express all recursively enumerable predicates. Hence, we can observe that the upper-bound for $\mathcal{L}_3$ is not given by its translation into Prolog but rather by a translation into well-known Description Logics. Since $\mathcal{L}_3$ allows (restricted) use of language features in $\mathcal{SHINO}$, we can conjecture that $\mathcal{L}_3$ has at most the same complexity as $\mathcal{SHINO}$, i.e. NEXPTIME. Even if we limit recursion and implement blocking (Buchheit et al., 1993) for the generated skolem functions, the blocking algorithm can still generate paths of exponential length before the actual blocking occurs. Obviously, we do not meet our goal of tractability anymore.

5.5.3. LP Translation

The recursive definition of $\phi_{\mathcal{LP}}$ (cf. Tables 5.3 and 5.2 on pages 122 and 120) shows that the $\phi_{\mathcal{LP}}$ translation into extended LP rules is linear in the depth of the class descriptions occurring in each axiom, i.e. the upper bound of the translation is

$$O((l + r) \times m)$$

where m is the number of axioms in $\mathcal{KB}_i^{DLP}$ and l (r) is the maximum depth of the class description on the left (right) hand side of any inclusion axiom in $\mathcal{T}_i^{DLP}$.

5.6. Conclusion

This chapter presented the DLP family of Description Logics. We introduced the syntax and constructors of each DLP variant and effected the semantics of these languages by a translation into various types of Logic Programs.

We compared the expressiveness of the languages with respect to the Web ontology languages presented in Chapter 3 on page 43 and showed from a practical perspective, that almost all parts of currently available Web ontologies can be expressed using the language.

We then showed how the typical DL reasoning problems can be reduced to operations on the logic program. In particular, we could identify that typical DL TBox reasoning tasks are expensive to handle, since their reduction involves updates on the logic program. On the other hand, we could also show that DL ABox reasoning tasks can be easily reduced to (multiple) LP queries.

Our analysis of the complexity of the DLP languages showed that their implementation in Datalog variants yields a polynomial of the fourth degree as an upper bound for the complexity of ABox reasoning tasks. We have to revoke our assumption, however, that the complexity of TBox reasoning problems is computationally simpler than their counter part in DLs, since we could only show an EXPTIME upper bound.

Moreover, we can conjecture from the possibility of non-termination when practically evaluating $\mathcal{L}_3$ in LP, that the language will not meet our initial goal of tractability due to the possibility of having to traverse paths of exponential length even when the blocking technique is applied.

Chapter 7 on page 173 will present our prototypical implementation, which can reuse both existing DL reasoners and existing LP engines for solving DLP reasoning problems. We then compare the performance of both embedded systems for the reasoning tasks available with the DLP languages in Chapter 8 on page 193. The evaluation is performed on synthetical knowledge bases, whose synthesis is based on a statistical analysis of the collection of Web ontologies, which we have met in this chapter.

The next chapter, however, will study how we can decrease the cost of reasoning with DLP further by materializing the implicit data that is derived by the DLP axioms.

6. Materialization

"Information ist nichts Festes und Fertiges,
sondern der Neuigkeitswert,
den wir aus Reizen ziehen. "
Georg Franck
Ökonomie der Aufmerksamkeit

This chapter extends (Volz et al., 2003f; Volz et al., 2003g) and presents a technique to incrementally maintain the materialization of Datalog programs. Materialization consists in precomputing and storing a set of implicit entailments, such that frequent and/or crucial queries to the reasoner can be solved more efficiently. The central problem that arises with materialization is the maintenance of a materialization when explicit axioms change, viz. the process of propagating changes in explicit axioms to the stored implicit entailments.

We can distinguish two types of changes in Description Logic Programs. Changes to the TBox will typically manifest themselves in changes to the rules of the compiled logic program, whereas changes to ABox assertions will typically lead to changes in facts. The incremental maintenance of the latter type of changes has been studied extensively in the deductive database context and we apply the technique proposed in (Staudt & Jarke, 1996) for our purpose. The former type of changes has, however, not been tackled before in the deductive database context and we extend the approach of (Staudt & Jarke, 1996) to deal with changes in rules. Our approach is not limited to Description Logic Programs but can be generally applied to arbitrary Datalog programs. We therefore briefly characterize at the end of this chapter how our approach could be applied to RDF rule languages presented in Section 3.5.2 on page 72.

The chapter is organized as follows: Section 6.1 introduces and motivates materialization for Semantic Web applications and discusses related work. Section 6.2 on page 149 presents the underlying principles which are applied to achieve incremental maintenance of a materialization. Section 6.3 on page 152 recapitulates the incremental maintenance algorithm presented in (Staudt & Jarke, 1996) and shows how this algorithm can be used to deal with changes in the ABox. Section 6.4 on page 159 extends this algorithm to deal with changing rules as they result from

changes in the TBox. Section 6.5 on page 165 summarizes our contribution and discusses further uses. In particular, Section 6.5.3 on page 167 sketches how the developed techniques can be applied in implementations of RDF rule languages.

6.1. Introduction

Germane to the idea of the Semantic Web are the capabilities to assert facts and to derive new facts from the asserted facts using the semantics specified by an ontology. Both current building blocks of the Semantic Web, RDF (Hayes, 2003) and OWL (McGuinness & van Harmelen, 2003), define how to assert facts and specify how new facts should be derived from stated facts.

6.1.1. Motivation

The derivation of information from asserted information is usually achieved at the time clients issue queries to inference engines. Situations where the query rate is high or the procedure to derive information is time consuming and complex lead to bad performance. Materialization can be used to increase the performance at query time and by making implicit information explicit. This avoids to recompute derived information for every query.

Materialization has been applied successfully in many applications where reading access to data is predominant. For example, data warehouses usually apply materialization techniques to make *online* analytical processing possible. Similarly, most Web portals maintain cached web pages to offer fast access to dynamically generated web pages.

We conjecture that reading access to ontologies is predominant in the Semantic Web and other ontology-based applications, hence materialization seems to be a promising technique for fast query processing.

Materialization is particularly promising for the currently predominant approach of aggregating information that is distributed into a central knowledge base (Decker et al., 1999; Heflin et al., 1999; Studer et al., 2002; Maedche et al., 2003). For example, the OntoWeb[1] Semantic portal (Spyns et al., 2002) employs a *syndicator* (cf. Figure 6.1), which regularly visits resources specified by community members and transfers the detected updates into a central knowledge base in a batch process, i.e. between updates the knowledge base remains unchanged for longer periods of time. The OntoWeb portal, however, answers queries from the

[1] `http://www.ontoweb.org/`.

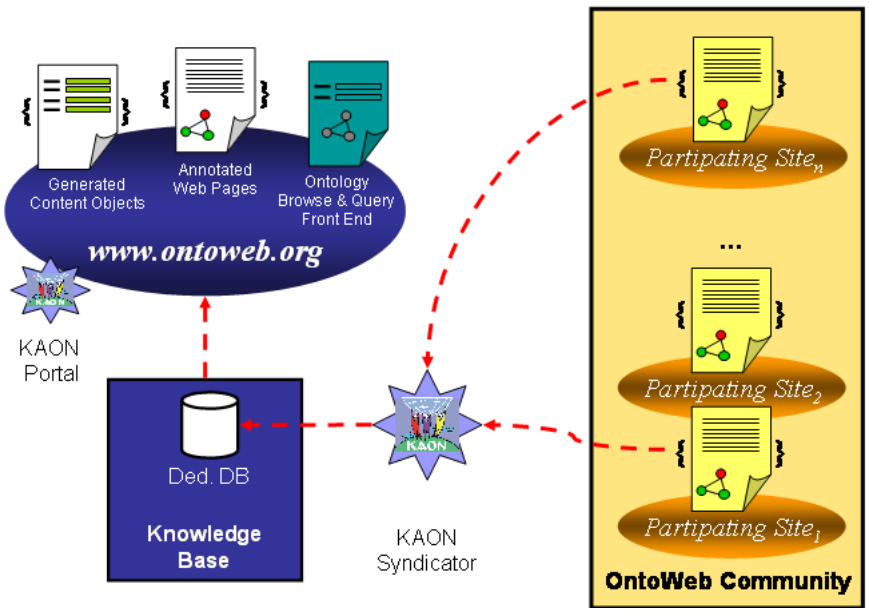


Figure 6.1.: OntoWeb Architecture

knowledge base whenever visitors browse the portal content. This is due to the fact that most queries are hard-coded into the definition of dynamic Web pages, which are generated for every request. In applications such as OntoWeb, materialization turns out to be a sine qua non.[2]

Central to materialization approaches is the issue of maintaining a materialization when changes occur. This issue can be handled by simply recomputing the whole materialization. However, as the computation of the materialization is often complex and time consuming, it is desirable to apply more efficient techniques in practise, i.e. to *incrementally* maintain a materialization.

6.1.2. Related Work

We can find related work in two areas: Firstly, incremental maintenance of materialized views in deductive databases. Secondly, truth maintenance systems in the

[2]Even though in OntoWeb, due to the unavailability of the solution developed in this chapter, the problem was approached by caching the Web pages through a proxy server.

Artificial Intelligence context.

Incremental Maintenance of Materialized Views Several algorithms have been devised for the incremental maintenance of materialized views in deductive databases. All of these approaches do not consider changes in the set of rules and differ in the techniques used to cope with changes in facts.

In order to cope with changing facts, (Apt & Pugin, 1987; Kuchenhoff, 1991) efficiently compute the Herbrand model of a stratified database after a database update. The proposed solution of (Apt & Pugin, 1987) uses sets of positive and negative dependencies that are maintained for all derived facts. This leads to low space efficiency and high cost for maintaining the dependencies. (Kuchenhoff, 1991) derives rules (so-called meta-programs) to compute the difference between consecutive database states for a stratified Datalog program. Some of the generated rules are not safe, making it impossible to implement the rules in Datalog engines. Additionally, duplicate derivations are not discarded in the algorithm.

(Gupta et al., 1993) presents the Delete and Re-Derive (DRed) algorithm, which is a procedural approach to view maintenance in Datalog with stratified negation. We will follow their principal approach for the computation of changes, in fact their procedural algorithm has been altered to a declarative algorithm (Staudt & Jarke, 1996) which we will extend.

The Propagation Filtration algorithm of (Harrison & Dietrich, 1992) is similar to the DRed algorithm, except that changes are propagated on a 'predicate by predicate' basis. Hence, it computes changes in one intensional predicate due to changes in one extensional predicate, and loops over all derived and extensional predicates to complete the maintenance procedure. In each step of the loop, the delete, re-derive and insert steps are executed. The algorithm ends up fragmenting computation and rederiving changed and deleted facts over and over again, i.e. it is less efficient than the DRed algorithm.

Truth Maintenance Systems (TMS) *Truth maintenance* [3] is an area of AI concerned with revising sets of beliefs and maintaining the truth in a reasoning system when new information alters existing information. A representation of beliefs and their dependencies is used to achieve the retraction of beliefs and to identify contradictions. For example, justification-based TMS (Doyle, 1981) uses a graph data structure where nodes are augmented with two fields indicating their belief status and supporting justification. When the belief status is changed, dependencies are propagated through the graph.

[3] also called *belief revision* or *reason maintenance*.

Making TMSs more efficient was a cottage industry in the late 1980s, with most of the attention focused on the Assumption-based TMS (ATMS) (de Kleer, 1986). The primary advantage of the ATMS is its ability to rapidly switch among many different contexts, which allows a simpler propagation of fact withdrawals, but comes at the cost of an exponential node-label updating process when facts are added. The main disadvantage of TMS is that the set of justifications (and nodes) grows monotonically as it is not allowed to retract a justification, but only disable information. The fact that the set of assumptions is always in flux introduces most of the complexity in the TMS algorithms. More recent work (e.g. (Nayak & Williams, 1998)) primarily tried to reduce the cost for incremental updates. However, the underlying principle of labelling does not change. To the best of our knowledge, there is no TMS, where the aggregation of all historic information is avoided, viz. facts are permanently removed from the system. Additionally the primary technique deployed in TMS (backtracking) does not fit well with the bottom-up computation that is usually applied in deductive databases. Recently, (Broekstra & Kampman, 2003) presented an adapted TMS algorithm for the incremental maintenance of RDF Schema entailments and uses the TMS labelling to track deductive dependencies between statements. This approach, however, is tailored to the RDF Schema language.

6.2. Maintenance Principles

This section discusses how the two main kinds of updates that have been mentioned in the introduction of the chapter, viz. updates to facts and rules, effect the materialization of an example knowledge base. Based on this discussion, we identify the main assumptions that underly the approaches for incremental maintenance presented in the subsequent sections.

As an example, we use the subset of the knowledge base presented in Table 2.5 on page 35 that is concerned with genealogical relationships between the different Bach family members. The relevant subset of the ABox is presented in Figure 6.2.

6.2.1. Updates to Facts

Since we expect that the historic data about the Bach family members in our knowledge base is unlikely to change, we choose to materialize the closure of the transitive ancestorOf property to speed up query processing. Figure 6.2 depicts an excerpt of the family tree of the Bach family, where the left-hand side of Figure 6.2 depicts the asserted property fillers. The right-hand side of the Figure depicts the transitive closure of the ancestorOf graph.

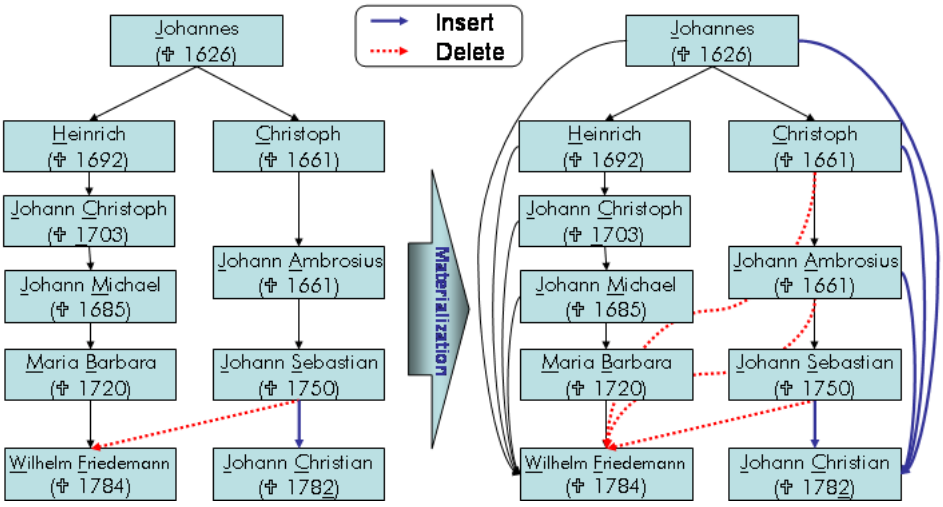


Figure 6.2.: Bach Family Tree Excerpt

Deletions Now assume that we have to revoke an asserted property filler, since a historian finds out that *Johann Sebastian* was not the father of *Wilhelm Friedemann*. Clearly, this has consequences to our materialization. For example, *Johann Ambrosius* is no longer an ancestor of *Wilhelm Friedemann*. However, *Johannes* is still an ancestor of *Wilhelm Friedemann*, since not only *Johann Sebastian* but also his cousin and first wife *Maria Barbara* are descendants of *Johannes*.

If we maintain the materialization of the graph depicted in Figure 6.2 ourselves, a natural and straightforward approach could be to proceed in two steps. We first mark all links leading from the *Wilhelm Friedemann* node to those nodes in the graph that possibly interact with the deleted link, viz. are also connected with *Johann Sebastian*. As a second step, we check whether the mark is correct by reconsidering whether the respective link could be generated on some other way through combining the links supported by the updated source graph. If a mark is determined to be correct, we can delete the appropriate link in our source graph. This two staged principle for deletion is common to most approaches for the incremental maintenance of materializations and is applied by the approach presented in Section 6.3.

Insertions Now assume that we assert that *Johann Sebastian* is the ancestor of another *Johann Christian*. Clearly, we can manually derive in the example graph that *Johann Christian* must be linked with those nodes that can possibly interact with the new link, viz. are also connected with *Johann Sebastian* in the updated source graph. All new links discovered in this way have to be added to the materialization.

6.2.2. Updates to Rules

A typical source of updates in Web ontologies is the change of TBox axioms, since ontologies have to evolve with their applications and react to changing application requirements (Maedche et al., 2002b; Maedche et al., 2003). Similarly, the advent of a rule layer in the Semantic Web will lead to changing rules. In the case of DLP, both situations, changing TBox axioms and changing rules, are actually equivalent since they manifest themselves as changes to the logic program $\mathcal{LP}$ that is created through the $\phi_{\mathcal{LP}}$ translation.

Axiom	*DL*	*DLP*
T_0	ANCESTOROF $\sqsubseteq$ INDYNASTY	INDYNASTY(x, y) : - ANCESTOROF(x, y)
T_1	ANCESTOROF$^+ \sqsubseteq$ ANCESTOROF	ANCESTOROF(x, y) : - ANCESTOROF(x, z), ANCESTOROF(z, y).

Table 6.1.: Example Knowledge Base

Let's assume that our TBox states that the transitive ancestorOf property is a specialization of the inDynasty property (which is not necessarily transitive), i.e. $\mathcal{T} = \{T_0, T_1\}$ (cf. Table 6.1). Let's additionally assume that our ABox only contains property fillers for ANCESTOROF, e.g. the tuples $\{(h, jc1), (j, h), (j, c) \ldots\}$, where each constant is the abbreviation for the name of an individual in the Bach family tree (cf. Figure 6.2). Clearly, the extension of both ANCESTOROF and INDYNASTY are equivalent in this particular knowledge base, since INDYNASTY has no own property fillers.

Manipulating the TBox $\mathcal{T}$ by deleting axiom T_0 leads to the situation that the extension of INDYNASTY is empty, since the derivations supported by the respective axiom are no longer supported.

Now assume that we add a new axiom T_2 to the old $\mathcal{T}$, i.e. $\mathcal{T} = \mathcal{T} \cup \{T_2\}$, that states that the property INDYNASTY is symmetric:

Axiom	*DL*	*DLP*
T_2	INDYNASTY$^- \sqsubseteq$ INDYNASTY	INDYNASTY(x, y) : - INDYNASTY(y, x).

Apparently, the new extension of INDYNASTY will now contain the tuple $(jc1, c)$ (among others), which is derived by the combination of the existing axioms and the new axiom.

Unlike the change of facts, we do not only have an interaction of particular (inserted or deleted) facts with existing facts, but also the interaction of (inserted or deleted)

rules with all other rules. In particular, we can observe that we need to consider all rules defining a predicate to determine the extension of the predicate.

The approach to maintenance presented in Section 6.4 will therefore recompute the extensions of all predicates, which are redefined by being the head of changed rules. We will, however, reuse the mechanisms of propagating the resulting fact changes to other predicates (and possibly back to the predicate in case of cycles) from the maintenance procedure for facts.

6.2.3. Differentiating Between Asserted and Entailed Information

The fundamental requirement for our approach to maintenance is the ability to distinguish entailed information from asserted information. This ability is required in order to propagate changes. The requirement also commonly arises in many ontology-based applications (Bechhofer et al., 2001), which often need to differentiate between asserted information and information that has been derived by making use of TBox axioms, e.g. when users attempt to change entailed information (Bechhofer et al., 2003b).

To achieve this differentiation, the $\phi_{\mathcal{LP}}$ translation has to be adapted such that all TBox axioms are translated into rules between purely intensional predicates C_{idb}, P_{idb}. ABox assertions, however, are stored in dedicated extensional predicates C_{edb}, P_{edb}. The connection between the intentional and the extensional database is made using simple rules that derive the initial (asserted) extension of the intensional predicates.

$$C_{idb}(x) \text{ :- } C_{edb}(x).$$
$$P_{idb}(x, y) \text{ :- } P_{edb}(x, y).$$

This storage architecture additionally allows - in principle - to connect the properties and classes of the ontology with arbitrary other predicates in the logic program. For example, most LP engines allow to populate predicates with queries issued to external databases. We could, hence, populate the ABox with data from relational databases which provides a simple form of data integration.

6.3. Maintaining Changing Facts

This section presents the maintenance of a materialization when facts change, viz. new tuples are added or removed from the extension of a predicate.

6.3.1. Approach

We reuse the declarative variant (Staudt & Jarke, 1996) of the delete and re-derive (DRed) algorithm proposed in (Gupta et al., 1993). DRed takes the three steps illustrated in Section 6.2.1 to incrementally maintain the materialization of a intensional database predicate:

1. *Overestimation of deletion:* Overestimates deletions by computing all direct consequences of a deletion.

2. *Rederivation:* Prunes those estimated deletions for which alternative derivations (via some other facts in the program) exist.

3. *Insertion:* Adds the new derivations that are consequences of insertions to extensional predicates.

The declarative version[4] of DRed maintains the materialization of a given predicate by means of a maintenance program. The maintenance program is rewritten from the original program using several rewriting patterns. The goal of the rewriting is the provision of a pair of maintenance predicates P^+ and P^- for every materialized predicate P, such that the extensions of P^+ and P^- contain the changes that are needed to maintain P after the maintenance program is evaluated on a given set of extensional insertions P^{Ins} and deletions P^{Del}.

The maintenance process is carried out as follows: First, we *setup* maintenance, i.e. the maintenance program is created for a given source program and the initial materialization of intensional predicates is computed.

Whenever extensional changes occur, the actual maintenance is carried out. In this step, we first put insertions (deletions) to an extensional predicate P_{edb} into the extension of the predicate P_{edb}^{Ins} (P_{edb}^{Del}). We then evaluate the maintenance program. For every intensional predicate P_{idb}, the required incremental changes, viz. insertions and deletions, can be found in the extension of P_{idb}^+ and P_{idb}^-. We use these changes to update the materialization of the intensional predicate P and update P_{edb} with the explicit changes P_{edb}^{Ins} and P_{edb}^{Del}, while the extensions of the later predicates are deleted.

[4] The benefit of reusing the declarative version of DRed with respect to the original (procedural) version is that it allows us to reuse generic logic databases for the evaluation of the maintenance program. This also motivates why we did not use the optimized version provided in (Staudt & Jarke, 1996), since the optimization requires logic databases to evaluate the maintenance program using the supplementary magic set technique, which is not used in all logic databases (e.g. XSB (Sagonas et al., 1994)).

6.3.2. Maintenance Rewritings

The maintenance of an intensional predicate $P \in \mathbf{P}_{idb}$ is achieved via seven maintenance predicates :

1. P^{Del} computes so-called deletion candidates, which are the aforementioned overestimation of facts that ought to be deleted from the materialization. For extensional predicates $P \in \mathbf{P}_{edb}$, P^{Del} contains explicitly what should be removed from the materialization.
2. P^{Ins} contains the facts that ought to be inserted into the materialization. For extensional predicates $P \in \mathbf{P}_{edb}$, P^{Ins} contains the explicit insertions that were asserted by the user.
3. P^{Red} stores those facts that are marked for deletion but have alternative derivations.
4. P^{New} describes the new state of the materialization after updates.
5. P^{+} computes the net insertions required to maintain the materialization.
6. P^{-} computes the net deletions required to maintain the materializaion.
7. P itself contains the (old) materialization.

New Materialization P^{New} captures the new materialization of an intensional database predicate P, which contains all old data that has not been deleted (N_1). Additionally, it contains re-derived data (N_2) and inserted data (N_3):

(N_1) $\quad P^{New} \text{ :- } P, \neg^* P^{Del}.$
(N_2) $\quad P^{New} \text{ :- } P^{Red}.$
(N_3) $\quad P^{New} \text{ :- } P^{Ins}.$

For every extensional database predicate P, we instantiate the rules (N_1 and N_3) to define an auxiliary predicate P^{New}, which is used in the rewritings for insertions and re-derivation of other (intensional) predicates.

Differentials The following differentials P^{+} and P^{-} compute positive and negative deltas, i.e. the changes that are necessary to incrementally maintain the stored materialization of an intensional predicate P:

$P^{+} \text{ :- } P^{Ins}, \neg^* P.$
$P^{-} \text{ :- } P^{Del}, \neg^* P^{Ins}, \neg^* P^{Red}.$

Deletion Candidates The deletion candidates P^{Del} are constituted by all possible combinations between deleted facts of a given body predicate and the remaining body predicates. Therefore, n deletion rules are created for every rule with n conjuncts in the body:

$$(D_i): \qquad P^{Del} \text{ :- } R_1, \ldots, R_{i-1}, R_i^{Del}, R_{i+1}, \ldots, R_n.$$

If R_i is an extensional predicate, R_i^{Del} contains those facts that are explicitly deleted from R_i. Otherwise, R_i^{Del} contains the aforementioned overestimation.

Re-derivations The re-derivations P^{Red} are computed by joining the new states of all body predicates with the deletion candidates:

$$(R): \qquad P^{Red} \text{ :- } P^{Del}, R_1^{New}, \ldots, R_n^{New}.$$

Insertions Insertions P^{Ins} are calculated by ordinary semi-naive rewriting, i.e. by constructing rules (I_i) that join the insertions into a body predicate with the new materializations of all other body predicates:

$$(I_i): \quad P^{Ins} \text{ :- } R_1^{New}, \ldots, R_{i-1}^{New}, R_i^{Ins}, R_{i+1}^{New}, \ldots, R_n^{New}.$$

If R_i is an extensional predicate, R_i^{Ins} contains those facts that are explicitly inserted into R_i.

A maintenance program is generated from a program $\mathcal{LP}$ through the application of generator functions. The rewriting generator functions are presented in Table 6.2 and make use of two auxiliary functions. The function $head : \mathcal{LP} \rightarrow \mathbf{P}_{idb}$ maps a rule to its rule head. Conversely, the function $rules : \mathbf{P}_{idb} \rightarrow \mathbf{R}$ maps rule heads to a set of rules $\mathbf{R}$, such that:

$$\forall P \in \mathbf{P}_{idb} : rules(P) = \{R \in \mathbf{R} | head(R) = P\}$$

For example, the function $\theta : R \rightarrow \mathbf{MR}$ rewrites a rule $R \in \mathcal{LP}$ into a set of maintenance rules $\mathbf{MR}$ by instantiating rewriting patterns for deletion θ^{Del}, insertion θ^{Ins} and rederivation θ^{Red}. By definition, θ maps every rule with n body literals into $2 * n + 1$ maintenance rules.

Definition 6.3.1 (Maintenance Program) *A maintenance program $\mathcal{LP}_M$ of a logic program $\mathcal{LP}$ over a signature $\Sigma = (C, F, \mathbf{P}, V, a)$, where $\mathbf{P} = \mathbf{P}_{idb} \cup \mathbf{P}_{edb}$ is a set of maintenance rules such that:*

Generator	Parameter	Rewriting Result
	Predicate	
θ_{idb}	$P \in \mathbf{P}_{idb}$	$\theta_{idb}^{New}(P) \cup \theta_{idb}^{Ins}(P) \cup \theta_{idb}^{Del}(P) \cup \theta_{idb}^{Red}(P)$
θ_{idb}^{New}	$P \in \mathbf{P}_{idb}$	$\{\theta_1^{New}(P)\} \cup \{\theta_2^{New}(P)\} \cup \{\theta_3^{New}(P)\}$
θ_{edb}^{New}	$P \in \mathbf{P}_{edb}$	$\{\theta_1^{New}(P)\} \cup \{\theta_3^{New}(P)\}$
θ_1^{New}	$P \in \mathbf{P}$	P^{New} :- $P, \neg^* P^{Del}$.
θ_2^{New}	$P \in \mathbf{P}_{idb}$	P^{New} :- P^{Red}.
θ_3^{New}	$P \in \mathbf{P}$	P^{New} :- P^{Ins}.
θ_{idb}^{+}	$P \in \mathbf{P}_{idb}$	P^{+} :- $P^{Ins}, \neg^* P$.
θ_{idb}^{-}	$P \in \mathbf{P}_{idb}$	P^{-} :- $P^{Del}, \neg^* P^{Ins}, \neg^* P^{Red}$.
θ_{idb}^{Ins}	$P \in \mathbf{P}_{idb}$	$\{\theta^{Ins}(r) \mid \forall r \in rules(P)\}$
θ_{idb}^{Del}	$P \in \mathbf{P}_{idb}$	$\{\theta^{Del}(r) \mid \forall r \in rules(P)\}$
θ_{idb}^{Red}	$P \in \mathbf{P}_{idb}$	$\{\theta^{Red}(r) \mid \forall r \in rules(P)\}$
	Rule	
θ	H :- $B_1, \ldots, B_n$.	$\{\theta^{Red}\} \cup \theta^{Del} \cup \theta^{Ins}$
θ^{Red}	H :- $B_1, \ldots, B_n$.	H^{Red} :- $H^{Del}, B_1^{New}, \ldots, B_n^{New}$.
θ^{Del}	H :- $B_1, \ldots, B_n$.	$\{H^{Del}$:- $B_1, \ldots, B_{i-1}, B_i^{Del}, B_{i+1}, \ldots, B_n.\}$
θ^{Ins}	H :- $B_1, \ldots, B_n$.	$\{H^{Ins}$:- $B_1^{New}, \ldots, B_{i-1}^{New}, B_i^{Ins}, B_{i+1}^{New}, \ldots, B_n^{New}.\}$

Table 6.2.: Rewriting Functions (derived from (Staudt & Jarke, 1996))

1. $\forall P \in \mathbf{P}_{idb} : \theta_{idb}(P) \in \mathcal{LP}_M$
2. $\forall P \in \mathbf{P}_{edb} : \theta_{edb}^{New}(P) \in \mathcal{LP}_M$

It is important to note that the rewritten rules do not only contribute to the computation of the differentials P^+ and P^-. The evaluation of the maintenance program also computes the set of implicit insertions and deletions that have to be propagated to the materialization of the predicate and to other predicates, which depend on the predicate through some rules.

Example 6.3.1 (Maintenance Rewritings) *Let us return to the Bach family tree example established in Section 6.2.1 and consider all edges between the different individuals depicted in Figure 6.2 on page 150 as fillers of the transitive property* ANCESTOROF *that was presented in the example knowledge base of Table 2.5 on page 35.*

The following logic program $\mathcal{LP}$ is constituted by the $\phi_{\mathcal{LP}}$ translation of the axiom that states the transitivity of the ANCESTOROF *property. The second rule implements the differentiation between asserted and entailed information, that was described in Section 6.2.3:*

(R_1) ANCESTOROF(x, z) :- ANCESTOROF(x, y), ANCESTOROF(y, z).
(R_2) ANCESTOROF(x, y) :- ANCESTOROF$_{edb}(x, y)$.

In the following we will use the abbreviation A for ANCESTOROF.

Since $\mathcal{LP}$ includes only one intensional (extensional) predicate $A(A_{edb})$, the generation of the maintenance program $\mathcal{LP}_M$ only involves to apply θ_{idb} to A and θ_{edb}^{New} to A_{edb}:

$$
\begin{array}{lll}
\theta_{edb}^{New}(A_{edb}) = & \{A_{edb}^{New}(x,y) \text{ :- } A_{edb}(x,y), \neg^* A_{edb}^{Del}(x,y). & (\theta_1^{New}(A_{edb})) \\
& A_{edb}^{New}(x,y) \text{ :- } A_{edb}^{Ins}(x,y).\} & (\theta_3^{New}(A_{edb})) \\
\theta_{idb}(A) = & \{A^{Del}(x,y) \text{ :- } A_{edb}^{Del}(x,y). & (\theta^{Del}(R_2)) \\
& A^{Red}(x,y) \text{ :- } A^{Del}(x,y), A_{edb}^{New}(x,y). & (\theta^{Red}(R_2)) \\
& A^{Ins}(x,y) \text{ :- } A_{edb}^{Ins}(x,y). & (\theta^{Ins}(R_2)) \\
& A^{New}(x,y) \text{ :- } A(x,y), \neg^* A^{Del}(x,y). & (\theta_1^{New}(A)) \\
& A^{New}(x,y) \text{ :- } A^{Red}(x,y). & (\theta_2^{New}(A)) \\
& A^{New}(x,y) \text{ :- } A^{Ins}(x,y). & (\theta_3^{New}(A)) \\
& A^{Del}(x,z) \text{ :- } A^{Del}(x,y), A(y,z). & (\theta^{Del}(R_1)) \\
& A^{Del}(x,z) \text{ :- } A(x,y), A^{Del}(y,z). & (\theta^{Del}(R_1)) \\
& A^{Red}(x,z) \text{ :- } A^{Del}(x,z), A^{New}(x,y), A^{New}(y,z). & (\theta^{Red}(R_1)) \\
& A^{Ins}(x,z) \text{ :- } A^{Ins}(x,y), A^{New}(y,z). & (\theta^{Ins}(R_1)) \\
& A^{Ins}(x,z) \text{ :- } A^{New}(x,y), A^{Ins}(y,z).\} & (\theta^{Ins}(R_1)) \\
\mathcal{LP}_M = & \theta(A) \cup \theta(A_{edb}) &
\end{array}
$$

The invocation of the θ_{edb}^{New} generator on A_{edb} initiates the invocation of the $(\theta_1^{New}(A_{edb}))$ and $(\theta_3^{New}(A_{edb}))$ generators and collects their results. Similarly, the invocation of the θ_{idb} generator on A leads to the invocation of $rules$ on A to retrieve the rules R_1, R_2 and the invocation of $\theta_1^{New}, \ldots, \theta^{Ins}(R_1)$.

6.3.2.1. DLP Maintenance Programs

If we consider the application of θ to the $\phi_{\mathcal{LP}}$ translation of $\mathcal{KB}_0^{DLP}$, we can observe from the structure of the rules that are generated by $\phi_{\mathcal{LP}}$ that the rewriting of each DL inclusion axiom creates the following number of maintenance rules:

$$
\begin{array}{lcl}
|\theta(\phi_{\mathcal{LP}}(C \sqsubseteq D))| & = & 3 \\
|\theta(\phi_{\mathcal{LP}}(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D))| & = & 2 * n + 1 \\
|\theta(\phi_{\mathcal{LP}}(C \sqsubseteq D_1 \sqcap \ldots \sqcap D_n))| & = & n * |\theta(\phi_{\mathcal{LP}}(C \sqsubseteq D_i))| \\
|\theta(\phi_{\mathcal{LP}}(D_1 \sqcup \ldots \sqcup D_n \sqsubseteq E))| & = & n * |\theta(\phi_{\mathcal{LP}}(D_i \sqsubseteq E))| \\
|\theta(\phi_{\mathcal{LP}}(C \sqsubseteq \forall R.D))| & = & |\theta(\phi_{\mathcal{LP}}(\exists R.C \sqsubseteq D))| = 5
\end{array}
$$

DL property transitivity is translated to five maintenance rules by applying θ to $\phi_{\mathcal{LP}}$. All other DL property axioms are translated to three maintenance rules by applying θ to $\phi_{\mathcal{LP}}$. θ is applied to all atomic classes and properties in $\mathcal{KB}_0^{DLP}$ as well as the auxiliary classes that are created by the structural transformation which is carried out during the preprocessing step.

6.3.2.2. RDF(S) Maintenance Programs

Since the 12 static Datalog rules for the single predicate-based axiomatization of RDF(S) (cf. Table 3.1 on page 59) contain 19 body predicates, the application of θ leads to the generation of 60 rules, namely 19 insertion rules, 19 deletion rules, 12 re-derivation rules, 5 maintenance rules for t^{New}, t^{+} and t^{-}, as well as 5 further rules to differentiate between entailments and assertions.

6.3.3. Evaluating Maintenance Programs

(Staudt & Jarke, 1995) show that the evaluation of the maintenance rules is a sound and complete procedure for computing the differentials between two database states when extensional update operations occur.

During the evaluation it is necessary to access the old state of a predicate. Bottom-up approaches to evaluation therefore require that all intensional relations involved in the computation are completely materialized, viz. the initial rules defining the predicates are not considered during the evaluation of the maintenance rules.

The maintenance rules for capturing the new database state contain negated predicates to express the algebraic set difference operation. Hence, even though the original rules are pure Datalog (without negation), a program with negation is generated. The rewriting transformation keeps the property of stratifiability, since newly introduced predicates do not occur in cycles with other negations. Hence, it is guaranteed that predicates can be partitioned into strata such that no two predicates in one stratum depend negatively on each other, i.e. predicates only occur negatively in rules that define predicates of a higher stratum. The evaluation can then proceed as usual stratum-by-stratum starting with the extensional predicates themselves.

Example 6.3.2 (Evaluating Maintenance Programs) *The direct links between members of the Bach family in Figure 6.2 on page 150 constitute the extension of A_{edb}, where we abbreviate the names of each individual by the first letters of their forenames:*

$$A_{edb} = \{(j,h),(j,c),(h,jc1),(jc1,jm),(jm,mb),(mb,wf),(js,wf),(ja,js),(c,ja)\}$$

Using the maintenance rewriting the materialization of A changes to A^{New} as follows, if $A^{Ins} = (js, jc2)$ is inserted and $A^{Del} = (js, wf)$ is deleted:

$$
\begin{array}{lcl}
A_{edb}^{Ins} & = & \{(jc, jc2)\} \\
A_{edb}^{Del} & = & \{(js, wf)\} \\
A_{edb}^{New} & = & A_{edb} \cup A_{edb}^{Ins} \setminus A_{edb}^{Del} \\
A^{Ins} & = & \{(js, jc2), (ja, jc2), (c, jc2), (j, jc2)\} \\
A^{Del} & = & \{(js, wf), (ja, wf), (c, wf), (j, wf)\} \\
A^{Red} & = & \{(j, wf)\} \\
A^{New} & = & (A \setminus A^{Del} \cup A^{Ins} \cup A^{Red}) \\
 & = & A \cup \{(js, jc2), (ja, jc2), (c, jc2), (j, jc2)\} \setminus \{(js, wf), (ja, wf), (c, wf)\} \\
A^{-} & = & \{(js, wf), (ja, wf), (c, wf)\} \\
A^{+} & = & \{(js, jc2), (ja, jc2), (c, jc2), (j, jc2)\}
\end{array}
$$

Since all maintenance rules of a given predicate have to be evaluated, an axiomatization of RDF(S) based on a single ternary predicate leads to complete recomputation in case of updates. We sketch an optimization for this case in Section 6.5.3 on page 167 which should result in more efficient evaluation for the single predicate axiomatization.

6.4. Maintaining Changing Rules

This section presents the maintenance of a materialization if the definition of rules changes, viz. rules that define a predicate are added or removed in the source program. We introduce two simple extensions to the rewriting-based approach presented in the previous section. Firstly, the materialization of predicates has to be maintained in the case of changes. Secondly, the maintenance programs have to be maintained such that additional rewritings are introduced for new rules and irrelevant rewritings are removed for deleted rules.

6.4.1. Approach

We illustrated in Section 6.2.2 on page 151 that every change in the rule set might cause changes in the extension of an intensional predicate P, with the consequence that the materialization of intensional predicates has to be updated. However, unlike in the case of changing extensions, both auxiliary predicates which capture the differences to update the materialization of some predicate $P \in \mathbf{P}_{idb}$, i.e. P^+ and P^- have empty extensions since no actual facts change.

Obviously, we can categorize the intensional predicates that are affected by a change in rules into two sets: *(I)* predicates that are directly affected, i.e. occur in the head of changed rules and *(II)* predicates that are indirectly affected, i.e. by depending on directly affected predicates through the rules in the program.

Our solution uses the existing maintenance rewriting for facts to propagate updates to the *indirectly affected* predicates. To achieve this, the maintenance computation for *directly affected* predicates is integrated into the maintenance program by redefining the auxiliary predicates that are used to propagate changes between predicates, i.e. P^{New}, P^{Ins} and P^{Del}.

6.4.2. Maintenance Rewriting

Let $\delta^+(\delta^-)$ be the set of rules which are inserted (deleted) from the logic program $\mathcal{LP}$. The reader may recall from the previous section that the function $head : \mathcal{LP} \rightarrow P_{idb}$ maps a rule to its rule head, and the function $rules : P_{idb} \rightarrow \mathcal{LP}$ maps rule heads to rules.

Definition 6.4.1 (Directly affected predicate) *An intensional predicate $p \in P_{idb}$ is a* directly affected *predicate, if $p \in \{head(r) | r \in \delta^+ \cup \delta^-\}$.*

Generator	Parameter	Rewriting Result
	Predicate	
ϑ	$P \in \mathbf{P}_{idb}$	$\{\vartheta^{Ins}_{idb}(P)\} \cup \{\vartheta^{Del}_{idb}(P)\} \cup \{\vartheta^{New}_{idb}(P)\}$
ϑ^{Ins}_{idb}	$P \in \mathbf{P}_{idb}$	P^{Ins} **:-** P^{New}.
ϑ^{Del}_{idb}	$P \in \mathbf{P}_{idb}$	P^{Del} **:-** P.
ϑ^{New}_{idb}	$P \in \mathbf{P}_{idb}$	$\{\vartheta^{New}(r) \mid \forall r \in rules(P)\}$
	Rule	
ϑ^{New}	H **:-** $B_1, \ldots, B_n$.	H^{New} **:-** $B_1^{New}, \ldots, B_n^{New}$.

Table 6.3.: Rewriting Functions

For every directly affected predicate P, the existing rules defining P^{New} in the maintenance program $\mathcal{LP}_M$ are deleted. Then, new rules axiomatize P^{New} using the (new) rule set that defines P in the updated original program. These rules are slightly adapted, such that references to any predicate P are altered to P^{New}, by instantiating the following rewriting pattern for all rules $R \in rules(P)$:

P^{New} **:-** $R_1^{New}, \ldots, R_n^{New}$.

The rewrite pattern simply states that the new state of the predicate P follows directly from the combination of the new states of the predicates R_i in the body of of all rules defining P in the changed source program.

All maintenance rules for calculating the insertions and deletions to P have to be removed from the maintenance program and are replaced by the following two static rules.

P^{Ins} :- P^{New}.
P^{Del} :- P.

The role of $P^{Ins}, P^{Del}, P^{New}$ is exactly the same as in the rewriting for facts, i.e. they propagate changes to dependent predicates. While P^{Ins} propagates the new state of a predicate as an insertion to all dependent predicates, P^{Del} propagates the old state of a predicate as a deletion to all dependent predicates. Figure 6.3 shows how the information flow in the maintenance program changes with respect to the rewriting of a rule $H(x)$:- $B(x)$. from the maintenance rewriting for fact changes *(a)* to the maintenance for rule changes *(b)*. The arrows to (from) nodes depict that the respective predicate possibly uses (is used by) some other predicate in the maintenance program.

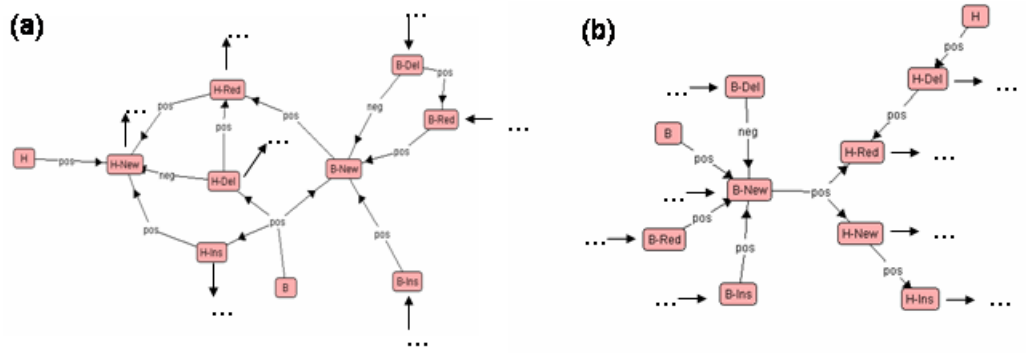


Figure 6.3.: Information Flow in Maintenance Programs: (a) Maintenance for Facts; (b) Maintenance for Rules

Table 6.3 defines new rewriting generator functions ϑ that implement the adapted rewriting that has been discussed above.

6.4.3. Evaluating Maintenance Programs

The evaluation of maintenance programs is now carried out in three steps:

- Update the maintenance rewriting $\mathcal{LP}_M$ of $\mathcal{LP}$ to incorporate the set of rules that are added (δ^+) or removed (δ^-).
- Evaluate the maintenance program $\mathcal{LP}_M$ and incrementally maintain all materialized predicates.

- Maintain the maintenance rewriting $\mathcal{LP}_M$ by changing rewritings back to the rewriting for facts.

Step 1 Step 1 is implemented by Algorithm 6.1 on page 163. This algorithm has three functions. Firstly, it replaces all maintenance rewritings for directly affected predicates with the new maintenance rewritings. Secondly, it alters the source program $\mathcal{LP}$ such that the set of updated rules is incorporated into $\mathcal{LP}$. Thirdly, it maintains auxiliary rewriting rules, viz. generates those rules for previously unknown intensional predicates and removes those rules if an intensional predicate no longer occurs in the source program.

Example 6.4.1 (Maintenance Rewritings for New Rule) *Let us return to the maintenance program $\mathcal{LP}_M$ established in Example 6.3.1 on page 156 and consider that rule R_3 is inserted into $\mathcal{LP} = \{R_1, R_2\}$, i.e. the new $\mathcal{LP}$ consists of the following rules after the application of Algorithm 6.1 on page 163:*

(R_1) $\text{ANCESTOROF}(x,z) \texttt{:-} \text{ANCESTOROF}(x,y), \text{ANCESTOROF}(y,z).$
(R_2) $\text{ANCESTOROF}(x,z) \texttt{:-} \text{ANCESTOROF}_{edb}(x,y).$
(R_3) $\text{INDYNASTY}(x,y) \texttt{:-} \text{ANCESTOROF}(x,y).$

Since $\delta^+ = R_3$ and $\delta^- = \emptyset$, the algorithm does not remove any rewriting rules from the maintenance program $\mathcal{LP}_M$ in this example, since none of the previously existing intensional axioms is directly affected. We have, however, to add the new maintenance rules for the directly affected predicate INDYNASTY*, which we will abbreviate as I in the following. The algorithm augments $\mathcal{LP}_M$ with the rules generated by the following calls to rewriting generators (in this order):*

$\theta^{Red}(R_3) = \quad I^{Red}(x,y) \texttt{:-} I^{Del}(x,y), \text{ANCESTOROF}^{New}(x,y).$
$\theta^{+}_{idb}(I) = \quad I^{+}(x,y) \texttt{:-} I^{Ins}(x,y), \neg^* I(x,y).$
$\theta^{-}_{idb}(I) = \quad I^{-}(x,y) \texttt{:-} I^{Del}(x,y), \neg^* I^{Ins}(x,y), \neg^* I^{Red}(x,y).$
$\vartheta^{New}(I) = \quad I^{New}(x,y) \texttt{:-} \text{ANCESTOROF}^{New}(x,y).$
$\vartheta^{Ins}(I) = \quad I^{Ins}(x,y) \texttt{:-} I^{New}(x,y).$
$\vartheta^{Del}(I) = \quad I^{Del}(x,y) \texttt{:-} I(x,y).$

The new state of I is now directly derived from the new state of A, which is calculated as part of the maintenance program. Hence, we can obtain the first materialization I just by evaluating the maintenance program.

Step 2 Step 2 evaluates the maintenance program as presented in Section 6.3.3.

Require:
δ^+ Set of inserted rules
δ^- Set of deleted rules
$\mathcal{LP}_M$ Maintenance program
Ensure:
Updated maintenance program $\mathcal{LP}_M$

removeMR = $\emptyset$ *// Collects maintenance rules to be removed*
addMR = $\emptyset$ *// Collects maintenance rules to be added*
affectedPred = $\emptyset$ *// Collects all affected predicates*
// Add new rewriting rules for added rules
for all $r \in (\delta^+ \setminus \delta^-)$ **do**
  addMR = $\theta^{Red}(r) \cup$ addMR
  p = head(r)
  affectedPred = $p \cup$ affectedPred
  // First rule defining a predicate ?
  if $p \notin \{head(r) \mid r \in \mathcal{LP}\}$ **then**
    addMR = $\theta^+_{idb}(p) \cup \theta^-_{idb} \cup$ addMR *// Need new auxiliary predicates*
  end if
  $\mathcal{LP} = \mathcal{LP} \cup r$
end for
// Add new rewriting rules for deleted rules
for all $r \in (\delta^- \setminus \delta^+)$ **do**
  p = head(r)
  affectedPred = $p \cup$ affectedPred
  // Last rule defining a predicate ?
  if $rules(p) \setminus \{r\} = \emptyset$ **then**
    removeMR = $\theta^+_{idb}(p) \cup \theta^-_{idb} \cup \theta^{Red}_{idb}(p) \cup$ removeMR
  end if
  $\mathcal{LP} = \mathcal{LP} \setminus r$
end for
// Replace rewriting rules for affected predicates
for all $p \in$ affectedPred **do**
  addMR = $\vartheta^{New}_{idb}(p) \cup \vartheta^{Ins}_{idb}(p) \cup \vartheta^{Del}_{idb}(p) \cup$ addMR
  removeMR = $\theta^{N}_{idb}(p) \cup \theta^{Ins}_{idb}(p) \cup \theta^{Del}_{idb}(p) \cup$ removeMR
end for
$LP_M = (LP_M \cup$ addMR $) \setminus$ removeMR

Algorithm 6.1: Updating Rules (Pre-Evaluation Algorithm)

Require:
δ^+ Set of inserted rules
δ^- Set of deleted rules
$\mathcal{LP}$ Original logic program
$\mathcal{LP}_M$ Maintenance program
Ensure:
Updated logic program $\mathcal{LP}$
Updated maintenance program $\mathcal{LP}_M$

removeMR = $\emptyset$
addMR = $\emptyset$
affectedPred = $\emptyset$
for all $r \in (\delta^+ \setminus \delta^-)$ **do**
  affectedPred = $head(r) \cup$ affectedPred
end for
for all $r \in (\delta^- \setminus \delta^+)$ **do**
  p = head(r)
  if $rules(p) \neq \emptyset$ **then**
    affectedPred = $p \cup$ affectedPred
  end if
end for
for all $p \in$ affectedPred **do**
  removeMR = $\vartheta_{idb}^{New}(p) \cup \vartheta_{idb}^{Ins}(p) \cup \vartheta_{idb}^{Del}(p) \cup$ removeMR
  addMR = $\theta_{idb}^{New}(p) \cup \theta_{idb}^{Ins}(p) \cup \theta_{idb}^{Del}(p) \cup$ addMR
end for
$LP_M = LP_M \cup$ addMR $\setminus$ removeMR

Algorithm 6.2: Updating Rules (Post-Evaluation Algorithm)

Step 3 Step 3 is implemented by Algorithm 6.2 on page 164. It essentially only undoes our special maintenance rewriting, i.e. it replaces the maintenance rewritings that have been generated by Algorithm 6.1 for directly affected predicates with the normal maintenance rewritings for facts.

Example 6.4.2 (Maintenance Rewritings for New Rule) *Algorithm 6.2 would remove the following maintenance rules from the maintenance program $\mathcal{LP}_M$:*

$$\begin{aligned}\vartheta^{New}(I) = &\quad I^{New}(x,y) \texttt{:-} \text{ANCESTOROF}^{New}(x,y).\\ \vartheta^{Ins}(I) = &\quad I^{Ins}(x,y) \texttt{:-} I^{New}(x,y).\\ \vartheta^{Del}(I) = &\quad I^{Del}(x,y) \texttt{:-} I(x,y).\end{aligned}$$

In parallel, the maintenance program would be extended with the rewritings generated by the rewriting generators that create the maintenance rewriting for facts ($\theta^{New}_{idb}(I), \theta^{Ins}_{idb}(I)$ and $\theta^{Del}_{idb}(I)$).

Since all maintenance rules for dealing with changes in rules are removed by Algorithm 6.2, we obtain the same maintenance program as if we would have completely regenerated the maintenance program for facts from the changed source program.

6.5. Conclusion

6.5.1. Contribution

We presented a technique for the incremental maintenance of materialized Datalog programs. Our technique can therefore be applied for those ontology languages, which can be axiomatized in Datalog, i.e. RDF Schema and $\mathcal{L}_{i\leq 2}$[5] as well as the Datalog-fragments of Semantic Web rule languages.

We contributed a novel solution to the challenge of updating a materialization incrementally when the rules of a Datalog program change, which has, to our best knowledge, not been addressed in the deductive database context[6].

In order to cope with changing rules, we applied a declarative, rewriting-based algorithm for the incremental maintenance of views (Staudt & Jarke, 1996) and introduced two novel techniques: Firstly, we extended the rewriting to deal with

[5] We cannot maintain function symbols other than constants, therefore our approach can not be used for $\mathcal{L}_3$.

[6] (Gupta et al., 1995) address the maintenance of views after redefinition for the relational data model.

changing rules. Secondly, we introduced two algorithms for the maintenance of the rewritten rules when the underlying source rules change.

Our solution has been completely implemented and evaluated. Section 7.5 on page 187 will report on our prototypical implementation. Section 8.6 on page 219 will present the results of our empirical analysis of the costs of incremental maintenance, which shows the feasibility of our solution.

The techniques proposed in this chapter are not specific to any ontology language, but can generally be used for the incremental maintenance of materialized Datalog programs. Due to this generic solution, future developments, e.g. for the rule layer of the Semantic Web, are likely to benefit from our technique as well.

Materialization is certainly not a panacea to all tractability problems. For example, one drawback is that it trades off required inferencing time against storage space and access time. In spite of this restriction, which remains to be assessed by more practical experience and cost models that are derived from those experiences, we conjecture that materialization as explained in this chapter will help to progress the Semantic Web and to build the large Semantic Web engines of tomorrow.

6.5.2. Further Uses

We can reuse the incremental maintenance of a materialization developed in this chapter, for several other purposes:

- *Integrity Constraint Checking*: Incremental maintenance can also be used as a fundamental technique in an implementation of integrity constraints on Semantic Web data, i.e. we can incrementally check the validity of a constraint by maintaining an empty view.

- *Continuous Queries*: (Liu et al., 1999) The auxiliary maintenance predicates P^+ and P^- can be used as a basis for implementing continuous queries or publish/subscribe systems, which are used to monitor a flow of data. This monitoring can use the extensions of P^+ and P^- as a basis for notification messages that are sent to the subscribers.

- *Interoperability with systems of limited inferencing capabilities*: We can use materialization to explicate data for clients that cannot entail information on their own. In particular, we can store materializations in relational databases which are agnostic about the semantics of the data but may be used for fast query answering.

Notation 3	`{ ?x :inDynasty ?y. }`
	`log:implies`
	`{ ?y :inDynasty ?x. }.`

Datalog	$T(x, \texttt{inDynasty}, y)\texttt{:-}T(y, \texttt{inDynasty}, x).$

Triple	`FORALL model @rules(model) {`
	`FORALL X,Y X[inDynasty->Y] <-`
	`Y[inDynasty->X]@model.}`

Datalog	$T(\texttt{model}, x, \texttt{inDynasty}, y)\texttt{:-}T(\texttt{model}, y, \texttt{inDynasty}, x).$

Table 6.4.: Datalog Translation of RDF Rule Languages

6.5.3. Materializing RDF Rules

We conclude this chapter, with a brief sketch of how the developed approach for incremental maintenance of Datalog programs can be applied in RDF rule engines.

6.5.3.1. Materializing RDF rule languages

An alternative to using $\mathcal{L}_0$ TBox axioms to state that INDYNASTY is a symmetric property, is the usage of either one of the RDF-based rule languages (cf. Section 3.5.2 on page 72), e.g. Triple or Notation 3.

The reader may recall that our incremental maintenance technique is not specific to DLP but generally works for Datalog programs. Hence, we can use it for any RDF rule base, that can be translated into Datalog programs (cf. Table 6.4 for an exemplary translation).

If an RDF rule system internally uses one single predicate within the rules, however, our technique for incrementally maintaining the materialization in case of changes is useless. The evaluation of the maintenance program then corresponds to a total recomputation, since all rules defining this predicate have to be evaluated.

In order to use our approach to materialization, more optimized data structures to represent an RDF graph have to be chosen, such that the part of the knowledge base which takes part in the evaluation can be limited.

6.5.3.2. Selection-based Optimization

We will briefly sketch a possible optimization, which we called *selection-based optimization* (Volz et al., 2003g). The optimization is based on the idea to split the extension of the RDF graph according to *split points*, which are given by constants that occur at a certain argument position of a predicate. Useful split points can be

derived from the vocabulary of an ontology or an ontology language such as RDF Schema. In case of arbitrary graph data, a useful split point can be frequently occurring constants, which can be easily determined using counting. The choice of a good split point, however, clearly depends on the application of the RDF rule base.

We can transform a Datalog program into an equivalent program that incorporates split points, if all references to a predicate P (in queries, facts and rules) where a split point occurs are replaced by appropriate split predicates.

In the following, we will assume that a split point is constituted by a constant c that is used as the i-th argument in the predicate P. To generate split predicates, we then split the extension of a predicate P_{edb} into several edb predicates of the form $P_{edb}^{c_i}(Var_1, Var_2, \ldots, Var_{i-1}, c, Var_{i+1}, Var_n)$ to store tuples based on equal constant values c in their i-th component.

Hence, instead of using a single extensional predicate P_{edb} for representing direct RDF assertions, the extensional database is split into several $P_{edb}^{c_i}$. Again, we can differentiate between asserted and derived information by introducing intensional predicates (views) for each component of the extension (i.e. rules of the form P^{c_i} :- $P_{edb}^{c_i}$).The complete predicate P can still be represented by means of an intensional predicate, which is axiomatized by a collection of rules that unify the individual split predicates: P :- P^{c_i}.

Example 6.5.1 *Returning to the triple based axiomatization (cf. Table 6.4 on page 167) of the N3 example, we can transform the program by introducing a split point $T^{inDynasty_2}$ for the* INDYNASTY *constant (when used as second argument in the ternary predicate T):*

- *We use two extensional predicates: $T_{edb}^{Rest}, T_{edb}^{inDynasty_2}$ to store the extension in two disjoint sets.*
- *We capture the intensional predicates and integrate the splits into a complete extension of T and rewrite the example such that split predicates are used instead of the full predicate:*

$$
\begin{array}{lll}
T^{Rest}(X,Y,Z) & \text{:-} & T_{edb}^{Rest}(X,Y,Z). \\
T^{inDynasty_2}(X,Y,Z) & \text{:-} & T_{edb}^{inDynasty_2}(X,Y,Z). \\
T(X,Y,Z) & \text{:-} & T^{Rest}(X,Y,Z). \\
T(X,Y,Z) & \text{:-} & T^{inDynasty_2}(X,Y,Z). \\
T^{inDynasty_2}(X,\mathit{inDynasty},Y) & \text{:-} & T^{inDynasty_2}(Y,\mathit{inDynasty},X).
\end{array}
$$

Any other rule that is inserted into the RDF rule base can be transformed into a set of rules, which use the available split predicates.

However, the maintenance of a materialized predicate $T^{inDynasty_2}$ can now be carried out by ignoring all non-relevant rules for T. Hence, the whole extension of T can be updated via the insert and delete maintenance rules that were presented in the previous sections, i.e. without using the complete database.

Part III.

Finale

7. Implementation

" Wir behalten von unseren Studien
am Ende doch nur das,
was wir praktisch anwenden. "
Johann-Wolfgang von Goethe

This chapter describes the prototypical implementation of DLP. The DLP prototype is a part of the KAON project (Bozsak et al., 2002), which is a joint effort of the knowledge management groups at AIFB and FZI and provides a general infrastructure for building ontology-based applications. The DLP implementation makes use of the OWL API (Bechhofer et al., 2003b), which is a Java-based API for accessing and manipulating OWL ontologies and is developed as part of the WonderWeb project[1]. Both KAON and the OWL API will be briefly described in this chapter.

The chapter is organized as follows. Section 7.1 motivates the need for an infrastructure for ontology-based applications. Section 7.2 on page 175 briefly introduces the KAON framework and discusses some components which are relevant for the DLP prototype. Section 7.3 on page 178 introduces the OWL API, which provides an API for accessing and manipulating OWL ontologies. Section 7.4 on page 180 presents the prototypical implementation of DLP and details the individual steps that have to be performed to obtain a logic program for a DL knowledge base. Section 7.5 on page 187 characterizes the implementation of materialization within the KAON datalog engine. Section 7.6 on page 188 discusses related work before we conclude in Section 7.7 on page 190 with a short summary of our contribution.

7.1. Introduction

Ontologies are increasingly being applied in complex applications, e.g. for Knowledge Management, E-Commerce, eLearning or information integration. In such

[1] Cf. `http://wonderweb.semanticweb.org/`, the project is funded as a shared-cost RTD under the European Commission Information Society Technologies (IST) programme (IST-2001-33052)

applications, ontologies serve various needs like storage or exchange of data corresponding to an ontology, ontology-based reasoning or ontology-based navigation. Building a complex ontology-based system usually encompasses several components, which can mostly be reused individually across the particular application at hand.

We therefore architectured the KAON tool suite as a family of reusable components for ontology-based applications. The fundamental idea of KAON is to avoid redundant implementation work by reuse of available components. The lifecycle of a KAON component, such as the DLP prototype, typically begins with a prototypical implementation of a particular research idea. This prototype is then validated through its usage in applications and other prototypes. The initial implementation is then improved and enhanced over time. The extension and improvement is usually done by several persons making it often difficult to attribute a particular component to an individual person. KAON is therefore the combined outcome of the efforts of many members of the knowledge management groups at AIFB and FZI.

The main focus of KAON has been to improve ontology management and reasoning by features standardized and widely adopted in the database community. Therefore our primary focus is not to provide a highly-expressive ontology model. KAON rather focuses to provide database features such as scalability, concurrency, persistence and transactions for enterprise-scale ontology-based applications. Some of these features are currently unique when compared with other efforts on ontology-based infrastructures. For example, KAON not only supports persistent storage of ontologies but also allows concurrent write access to ontologies by allowing transactional updates. The most important requirement for KAON is to ensure that the performance of accessing ontology-based data is sufficient for realistic applications. We therefore follow a conservative and incremental strategy with respect to the reasoning required to support an ontology language. We departed from a simple ontology language that largely rectified the semantics of RDF Schema (Bozsak et al., 2002). We then considered a more expressive language in (Motik et al., 2002a), which provided property characteristics. Currently, we can support the DLP family of languages presented in this thesis. Each of the languages have been accompanied with software components that allow to access and manipulate ontologies in the respective languages. The API for DLP allows to access and manipulate the whole OWL DL language (Bechhofer et al., 2003b) and has been developed[2] jointly with the University of Manchester in context of the WonderWeb project.

[2]Outside of KAON as an open-source software project on its own.

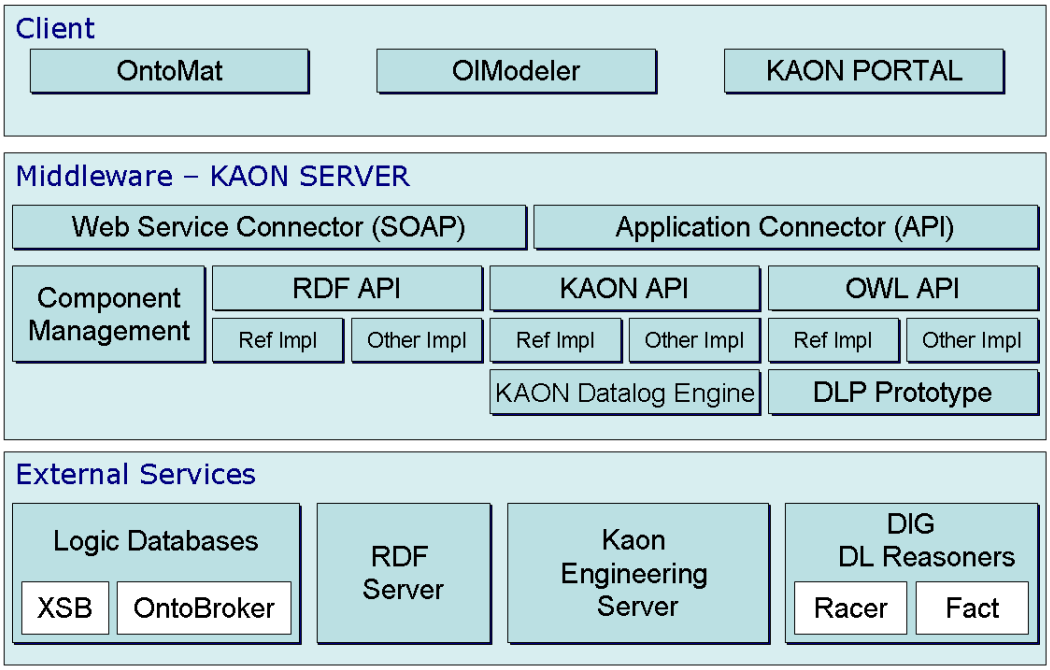


Figure 7.1.: KAON Architecture

7.2. KAON

This section gives a brief introduction to the Karlsruhe Ontology and Semantic Web Tool Suite (KAON). KAON provides the structural environment for the development of the DLP prototype and contributes several components used within the DLP prototype. We will present those parts of the KAON architecture that are relevant to DLP.

KAON builds on experiences from previous developments and projects dealing with semantics-based applications in the areas of E-Commerce, Knowledge Management and Web Portals. KAON is an open-source software project that is implemented in Java. KAON provides a family of specialized tools for engineering, discovery, management and presentation of ontologies and metadata.

The KAON architecture is depicted in Figure 7.1 and consists of three layers, i.e. the client layer, the middleware layer and the external services layer.

7.2.1. Client Applications

We can distinguish several client applications:

- **OntoMat** (Handschuh & Staab, 2002)- desktop application framework for annotation tools.
- **OIModeler** - desktop application framework for ontology editors.
- **KAON PORTAL** - Web portal framework for constructing ontology-based Web portals.

Client applications may either connect to or directly embed components residing in the middleware layer. This may be the OWL API component, which provides programmatic access to OWL ontologies and knowledge bases, or the RDF API to access RDF data sets. The main functionality of client applications is devoted to their specific purpose, e.g. authoring ontology-based annotations or HTML rendering of ontologies. All clients require, however, common functionality such as the representation of views and controllers for models realized by the ontology APIs.

7.2.2. Middleware Layer

The primary role of the middleware layer is to provide an abstraction for ontology and data access. The middleware layer is implemented through the KAON SERVER (Volz et al., 2003c), which is a generic application server that allows to manage the components needed for ontology-based applications. KAON SERVER also allows the composition of components with functionality provided by external services such as reasoning engines. The composition is provided by a dedicated component management module which supports the dynamic instantiation and localization of appropriate components and delegates requests to components.

The functionality required for a particular application can also be provided by external services, thus it is possible to delegate requests to those services. The interested reader may refer to (Motik et al., 2002c) for more details.

Data access is realized by three programmatic APIs:

- OWL API - offering access to ontologies and knowledge bases in the OWL ontology language.
- KAON API - offering access to ontologies and knowledge bases in the KAON ontology language.

- RDF API - offering access to RDF data.

All three APIs provide interfaces for the programmatic access to ontologies and instances independent of their physical storage. Access to a particular physical store is provided by implementations of the interface. All APIs have one reference implementation, which allows to access data stored in files (or in URLs on the Web). The KAON API and the RDF API have alternative implementations, which allow to access data hosted by dedicated servers such as the RDF SERVER and the KAON ENGINEERING SERVER. The KAON and OWL APIs can ensure the consistency of the ontology and adhere to the formal semantics of the respective language. The APIs also support the composition of distributed ontologies by means of inclusion. Some implementations of the KAON API additionally support concurrent access and transactional processing.

The RDF API provides programmatic access to RDF models. It features proprietary means for modularization, a fast and efficient RDF parser and serializer as well as transactional access.

The KAON Datalog engine is a main-memory based logic database, which implements the Magic Set technique for optimized reasoning. It is used to provide reasoning services for the KAON API and can be used to provide reasoning services for the OWL API via the DLP prototype described in Section 7.4.

7.2.3. External Services Layer

This layer has several roles. Firstly, it offers access to physical data stores such as databases, file systems or the network. Secondly, it groups separate, external software entities such as reasoning engines. These services are accessed through connectors, which are managed by the KAON SERVER.

KAON SERVER currently allows to connect to DL engines, which support the DIG (Bechhofer et al., 1999) interface and several rule-based inference engines such as OntoBroker (Decker et al., 1999) and XSB (Sagonas et al., 1994).

KAON itself provides two external services, namely an RDF SERVER, which is able to store RDF data and allows concurrent, transactional access to RDF data, and an ENGINEERING SERVER, which is able to store KAON ontologies and allows concurrent and transactional access to KAON ontologies. The latter uses an optimized database schema to increase performance in settings where an abstraction from RDF can be made, e.g. in ontology engineering scenarios.

Both services are intended to handle database content and may be used transparently through the KAON and RDF API. Non-RDF data sources may be accessed

using other implementations of the KAON or OWL API, for example to provide a connection to ABoxes that are stored in relational databases.

Future versions of the KAON Tool Suite may provide further components. The interested reader is therefore invited to visit the KAON Web site[3] to check for current updates.

7.3. OWL API

This section briefly characterizes the OWL API (Bechhofer et al., 2003b), which is used within the DLP prototype to read OWL and DLP ontologies. The OWL API presents a highly reusable component for the construction of different ontology-based applications such as editors, annotation tools or query agents. Besides enabling them to "talk the same language", it ensures that they share underlying assumptions about the way that information is presented and represented. Thus a cornerstone to the successful implementation and delivery of the Semantic Web, namely the interoperability of applications, is achieved.

7.3.1. Aspects of Functionality

Since different classes of ontology-based applications require different aspects of functionality, the OWL API covers several aspects beyond the representation of OWL data through its implementation. To avoid a single monolithic API, the API is split into several packages, which support the following functionality:

- *Serializing:* Producing a concrete syntactic form of OWL (for example as RDF statements or using the OWL abstract syntax) from some internal data structure or representation;
- *Modelling:* Providing data structures that can represent/encode OWL documents. This representation basically introduces an interface for every OWL construct and appropriate generalizations. These generalizatons facilitate access to OWL information, e.g. users can ask for all restrictions independent of the particular variant of restriction;
- *Parsing:* Taking a concrete representation of an OWL document (e.g. an RDF-XML serialization of an OWL document) and building some internal representation that corresponds to that document;

[3] `http://kaon.semanticweb.org/`.

- *Manipulating:* Providing representation along with mechanisms for manipulation of OWL ontologies and knowledge bases;
- *Inferencing:* Providing a representation that allows to access entailed information and to check the consistency of the knowledge base by means of external reasoners[4].

Particular applications that make use of the API are not expected to require all packages. For example, a format/syntax translator acts as a client of the API and requires the ability to parse, to represent the results of the parsing in some way and then to serialize. A simple ontology editor would also require manipulation capabilities to allow construction and editing of ontologies (i.e. definitions of classes, properties and so on). A simple editor, however, may not need any functionality relating to semantics or inference, e.g. the facility for checking the consistency of class definitions or whether subsumption relationships can be inferred. Alternatively, an application that simply deploys an ontology to client applications may not require any functionality that supports serialization, manipulation of the ontology, but needs support to access the ontology model and its entailments. For example, a reasoner would support inference, but need not be concerned with issues relating to serialization and parsing.

7.3.2. Modelling

As we have seen in Section 3.4 on page 60, the OWL syntax allows to define classes using both a definitional style, i.e. the use of class definitions, and through axioms, i.e. the use of class inclusion axioms and class equivalence axioms. From a semantical perspective, both ways of defining an ontology are equivalent. Both definitions, however, convey slightly different ways of modelling the world in terms of how the creator of the ontology thinks that things fit together. The OWL API therefore not only ensures that we capture the correct semantics of the ontology, but also the semiotics (Euzenat, 2000). While the API faithfully represents each alternative used for definitions, it also allows to normalize all definitions into their corresponding axioms. This functionality is especially useful for reasoners, which are mainly concerned with the axiomatic view. For example, the DLP prototype only translates axioms into LP rules.

The API separates asserted from inferred data such as proposed in Section 6.2.3 on page 152. This separation is not only important for the purpose of materialization but generally important for ontology editors such as OilEd (Bechhofer et al.,

[4] Via the DIG interface.

2001). Clearly, we could simply add inferred information to the asserted information, since this does not change the underlying semantics of the ontology (as they are already inferred, we are simply adding redundant information). However, the experience with OilEd (Bechhofer et al., 2001) shows that this leads to confusion, in particular when users want to further edit the amended ontology, i.e. want to delete inferred information (that keeps reappearing since it can be inferred).

7.3.3. Inferencing

The OWL API itself does not provide any OWL inference capabilities, but rather reuses external inference engines for that purpose. The OWL specification, however, includes a detailed description of the semantics of the language, defines what entailment precisely means with respect to OWL ontologies and provides formal descriptions of properties such as consistency.

The implementation therefore provides a standard interface that establishes abstract methods which allow to access entailed information and to check the consistency of the ontology. These abstract methods can then be implemented by querying OWL reasoners. The definition of a standard interface ensures that particular applications can exchange reasoners at their leisure, since multiple reasoners conform to the same interface.

Of course, providing abstract method signatures does not go all the way to advertising the functionality of an implementation – there is no guarantee that a component implementing the `inference` interface necessarily implements the semantics correctly. However, signatures go some way towards providing an expectation of the operations that are being supported. Collections of test data (such as the OWL Test Cases (Carroll & De Roo, 2003)) can allow systematic testing and a level of confidence as to whether the implementation is, in fact, performing correctly.

7.4. DLP Prototype

This section describes the DLP prototype which is currently nothing more than a simple compiler from DL knowledge bases to corresponding logic programs. This section details the different steps that are carried out as part of the compilation.

7.4.1. Prototype Architecture

We implemented the DLP prototype as a simple compiler which accepts a given OWL knowledge base and generates the corresponding logic program.

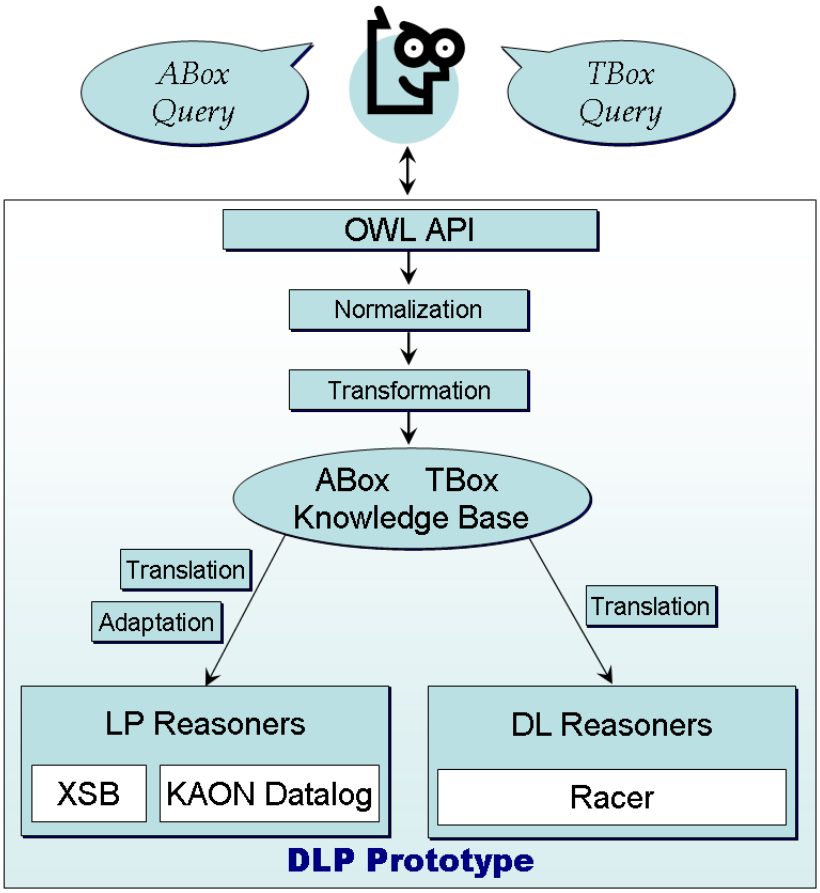


Figure 7.2.: DLP Prototype

Within the DLP prototype we use several (external) components. The RDF API delivered by the KAON project (Bozsak et al., 2002) is used for parsing RDF data. The OWL API (Bechhofer et al., 2003b) is used for parsing OWL and representing OWL.

On top of the compiler, we also provide three methods to answer simple $\mathcal{QDL}$ queries for the checking subsumption between two classes, allow to retrieve the individuals that instantiate a certain class and property fillers. The evaluation presented in Section 8.4 will compare the response time of such queries between logic databases and Racer (Haarslev & Moller, 2001), the only publicly available DL reasoning engine which supports ABoxes.

The DLP prototype can currently access the following two logic databases for answering the above mentioned $\mathcal{QDL}$ queries:

- *XSB (Sagonas et al., 1994)* is a Prolog engine that employs tabled resolution (Swift & Warren, 1994).
- *KAON Datalog* is an emerging (disjunctive) Datalog engine developed as part of the KAON project (Bozsak et al., 2002).

7.4.2. Compilation

The compilation departs from a given DLP knowledge base which is stored in OWL/RDF documents. The parsing and validation of OWL/RDF is provided by the OWL API (Bechhofer et al., 2003b) which then models the knowledge base through objects.

We then compile the logic program through the following sequence of processing steps:

1. All class and property definitions are normalized to the corresponding DL axioms.
2. All class descriptions are normalized and simplified using the rewriting algorithm described in Figure 4.2 on page 85.
3. All class axioms are normalized using the rewritings presented in Section 4.3.2 on page 86.
4. All axioms are transformed into a new set of axioms using the structural transformation which decomposes all nested class descriptions.
5. We then check, whether every axiom is a DLP axiom. If this is not the case, the respective axiom is not translated[5].
6. All DLP axioms are translated into LP rules using the variant of the $\phi_{\mathcal{LP}}$ translation described in the subsequent Section 7.4.3.
7. The resulting logic program is adapted to the LP variant supported by the target engine (cf. Section 7.4.4 on page 185).

The resulting logic program can then be saved in the format of the LP engines listed above.

[5] Users are informed of the untranslated axioms

7.4.3. Translation and Transformation

The translation function $\phi_{\mathcal{LP}}$ as defined in Table 5.3 on page 122 outputs extended LP rules which can only be used in LP engines after an a-posteriori transformation.

As we could see in Section 5.2.3 on page 122, four basic types of transformations are needed which are constituted by conjunction in the head, implication in the head, disjunction in the body and negation in the head of extended LP rules. In these cases, we can obtain plain LP rules by the Lloyd-Topor transformations (Lloyd, 1987). These transformations create several plain LP rules out of one extended rule. We can however observe, that the individual constituents of extended rules are still present and were simply relocated to some other place in the new rules.

The new translation function $\varphi_{\mathcal{LP}}$ (cf. Table 7.1 on page 184) carries the individual constituents of rules through the translation process. If a disjunction occurs in the body, the translation function simply creates two rules with equivalent heads. If an implication occurs in the head of the rule, the body of the implication is moved to the body of the rule. If a conjunction occurs in the head of the rule, two new rules with the same body are created.

The system has to remember the context in which $\varphi_{\mathcal{LP}}$ is applied. This context is represented by two new variables which carry rule fragments that are designated to become part of the head (H) and the body (B) of the rule once the translation is finished. Consider for example an LP rule where $C \sqcup D$ occurs in the body of the rule, viz. the rule has the form H :- $(C \sqcup D), B.$, where H is the head of the rule and B is an arbitrary body fragment. Our new translation function $\varphi^L_{\mathcal{LP}}((C \sqcup D), x)$ will replicate H and B into two rules (cf. Table 7.1 on page 184), which have the same head H and both contain the fragment B in their bodies. However, one rule will contain the translation $\varphi^L_{\mathcal{LP}}(C, x)$ in the body, whereas the other rule will contain the translation $\varphi^L_{\mathcal{LP}}(D, x)$.

This means that the translation process is no longer carried out recursively but incrementally. Every inclusion axiom in the DL knowledge base is pushed on a stack. Every translation step pops one axiom from the stack and expands one single class description of the axiom. Partially expanded axioms are pushed back on the stack, while completely expanded axioms are appended to the LP program. The translation process stops when the stack is empty. The following example illustrates the translation:

Example 7.4.1 *Consider the following inclusion axiom:*

$$A \sqcup B \sqsubseteq C \sqcap D$$

The axiom is translated into LP rules as follows. Since no normalization and simplification is possible, only structural transformation is applied and results in the following set of DL

$\mathcal{LP}_0 = \text{Datalog}$

$$\varphi_{\mathcal{LP}}(C \equiv D) \longrightarrow \begin{cases} \varphi_{\mathcal{LP}}(C \sqsubseteq D) \\ \varphi_{\mathcal{LP}}(D \sqsubseteq C) \end{cases}$$

$$\varphi_{\mathcal{LP}}(C \sqsubseteq D) \longrightarrow \varphi^R_{\mathcal{LP}}(D, x) \text{ :- } \varphi^L_{\mathcal{LP}}(C, x).$$

$$\begin{array}{ll}
\varphi^R_{\mathcal{LP}}(A, x) \text{ :- } B. & \longrightarrow \quad A(x) \text{ :- } B. \\
\varphi^R_{\mathcal{LP}}(\exists R.\{i\}, x) \text{ :- } B. & \longrightarrow \quad R(x, i) \text{ :- } B. \\
\varphi^R_{\mathcal{LP}}(C \sqcap D, x) \text{ :- } B. & \longrightarrow \quad \begin{cases} \varphi^R_{\mathcal{LP}}(C, x) \text{ :- } B. \\ \varphi^R_{\mathcal{LP}}(D, x) \text{ :- } B. \end{cases} \\
\varphi^R_{\mathcal{LP}}(\forall R.C, x) \text{ :- } B. & \longrightarrow \quad \varphi^R_{\mathcal{LP}}(C, y_i) \text{ :- } R(x, y_i), B. \\
H \text{ :- } \varphi^L_{\mathcal{LP}}(\exists R.\{i\}, x), B. & \longrightarrow \quad H \text{ :- } R(x, i), B. \\
H \text{ :- } \varphi^L_{\mathcal{LP}}(A, x), B. & \longrightarrow \quad H \text{ :- } A(x), B. \\
H \text{ :- } \varphi^L_{\mathcal{LP}}((\exists R.C), x), B. & \longrightarrow \quad H \text{ :- } R(x, y_i), C(y_i), B. \\
H \text{ :- } \varphi^L_{\mathcal{LP}}((C \sqcap D), x), B. & \longrightarrow \quad H \text{ :- } \varphi^L_{\mathcal{LP}}(C, x), \varphi^L_{\mathcal{LP}}(C, x), B. \\
H \text{ :- } \varphi^L_{\mathcal{LP}}((C \sqcup D), x), B. & \longrightarrow \quad \begin{cases} H \text{ :- } \varphi^L_{\mathcal{LP}}(C, x), B. \\ H \text{ :- } \varphi^L_{\mathcal{LP}}(D, x), B. \end{cases}
\end{array}$$

$\mathcal{LP}_1 = \text{Datalog(=)}$

$$\begin{array}{ll}
\varphi^R_{\mathcal{LP}}(\leq 1\, R, x) \text{ :- } B. & \longrightarrow \quad y_1 = y_2 \text{ :- } R(x, y_1), R(x, y_2), B. \\
\varphi^R_{\mathcal{LP}}(\{i\}, x) \text{ :- } B. & \longrightarrow \quad (x = i) \text{ :- } B. \\
H \text{ :- } \varphi^L_{\mathcal{LP}}(\{i_1, \ldots, i_n\}, x), B. & \longrightarrow \quad \begin{cases} H \text{ :- } B, (x = i_1). \\ \ldots \\ H \text{ :- } B, (x = i_n). \end{cases} \\
H \text{ :- } \varphi^L_{\mathcal{LP}}(\top, x), B. & \longrightarrow \quad H \text{ :- } (x = x), B. \\
\varphi^R_{\mathcal{LP}}(\top, x) \text{ :- } B. & \longrightarrow \quad (x = x) \text{ :- } B.
\end{array}$$

$\mathcal{LP}_2 = \text{Datalog(=,IC)}$

$$\begin{array}{ll}
\varphi^R_{\mathcal{LP}}((\neg A), x) \text{ :- } B. & \longrightarrow \quad \text{:- } B, A(x). \\
\varphi^R_{\mathcal{LP}}(\bot, x) \text{ :- } B. & \longrightarrow \quad \text{:- } B, (x = x).
\end{array}$$

$\mathcal{LP}_3 = \text{Prolog(=,IC)}$

$$\begin{array}{ll}
H \text{ :- } \varphi^R_{\mathcal{LP}}(\exists R.C, x), B. & \longrightarrow \quad \begin{cases} C(f_i(x) \text{ :- } B. \\ R(x, f_i(x) \text{ :- } B. \end{cases} \\
\varphi^R_{\mathcal{LP}}(\geq n\, R, x) \text{ :- } B. & \longrightarrow \quad \begin{cases} \{f_i(x) \neq f_j(x) \text{ :- } B. | 1 \leq i < j \leq n\} \cup \\ \{R(x, f_i(x)) \text{ :- } B. | 1 \leq i \leq n\} \end{cases}
\end{array}$$

Table 7.1.: Translation with Integrated Transformation

axioms:

$$A \sqcup B \sqsubseteq E$$
$$F \sqsubseteq C \sqcap D$$
$$E \sqsubseteq F$$

The application of $\varphi_{\mathcal{LP}}$ then proceeds as follows:

$LPProgram$	$Stack$
$\emptyset$	$\varphi^R_{\mathcal{LP}}(E,x)\text{:-}\varphi^L_{\mathcal{LP}}(A \sqcup B,x).$ $\varphi^R_{\mathcal{LP}}(C \sqcap D,x)\text{:-}\varphi^L_{\mathcal{LP}}(F,x).$ $\varphi^R_{\mathcal{LP}}(F,x)\text{:-}\varphi^L_{\mathcal{LP}}(E,x).$
$F(x)\text{:-}E(x).$	$\varphi^R_{\mathcal{LP}}(E,x)\text{:-}\varphi^L_{\mathcal{LP}}(A \sqcup B,x).$ $\varphi^R_{\mathcal{LP}}(C \sqcap D,x)\text{:-}\varphi^L_{\mathcal{LP}}(F,x).$
$F(x)\text{:-}E(x).$	$\varphi^R_{\mathcal{LP}}(E,x)\text{:-}\varphi^L_{\mathcal{LP}}(A \sqcup B,x).$ $C(x)\text{:-}\varphi^L_{\mathcal{LP}}(F,x).$ $D(x)\text{:-}\varphi^L_{\mathcal{LP}}(F,x).$
$F(x)\text{:-}E(x).$ $D(x)\text{:-}F(x).$	$\varphi^R_{\mathcal{LP}}(E,x)\text{:-}\varphi^L_{\mathcal{LP}}(A \sqcup B,x).$ $C(x)\text{:-}\varphi^L_{\mathcal{LP}}(F,x).$
$F(x)\text{:-}E(x).$ $D(x)\text{:-}F(x).$ $C(x)\text{:-}F(x).$	$\varphi^R_{\mathcal{LP}}(E,x)\text{:-}\varphi^L_{\mathcal{LP}}(A \sqcup B,x).$
$F(x)\text{:-}E(x).$ $D(x)\text{:-}F(x).$ $C(x)\text{:-}F(x).$	$\varphi^R_{\mathcal{LP}}(E,x)\text{:-}\varphi^L_{\mathcal{LP}}(A,x).$ $\varphi^R_{\mathcal{LP}}(E,x)\text{:-}\varphi^L_{\mathcal{LP}}(B,x).$
$F(x)\text{:-}E(x).$ $D(x)\text{:-}F(x).$ $C(x)\text{:-}F(x).$ $E(x)\text{:-}B(x).$	$\varphi^R_{\mathcal{LP}}(E,x)\text{:-}\varphi^L_{\mathcal{LP}}(A,x).$
$F(x)\text{:-}E(x).$ $D(x)\text{:-}F(x).$ $C(x)\text{:-}F(x).$ $E(x)\text{:-}B(x).$ $E(x)\text{:-}A(x).$	$\emptyset$

7.4.4. Variants of Logic Databases

Most logic databases vary in their support for particular primitives. This section shows how equality, inequality and integrity constraints can be simulated with the core Datalog language. We can resort to these simulations, if a given logic database does not (fully) provide those given primitives.

7.4.4.1. Equality

Since equality has almost the same universality like logical connectives and quantifiers, most LP engines provide a built-in equality predicate which we can use in our translation. Sometimes, however, the equality predicate is missing or its use is restricted. For example, XSB does not allow[6] to assign equality in the head of rules.

In these cases we have to provide a dedicated predicate that plays the role of equality. We further have to provide a correct axiomatization of the predicate using the equivalence theory of Definition 2.3.13 on page 24.

The substitutivity component of equivalence requires that all intentional predicates are described by further rules which incorporate the presence of equality:

$$\begin{gathered} C(X) \texttt{:-} C(Y), X = Y. \\ P(X,Y) \texttt{:-} P(X,Z), (Y = Z). \\ P(Z,Y) \texttt{:-} P(X,Y), (X = Z). \end{gathered}$$

If the LP engine at hand disallows unsafe rules, we have to drop the (unsafe) reflexivity rule from the equivalence theory. We can, however, still capture the reflexivity. One possible alternative is to capture the Herbrand universe explicitly in a predicate $HU(x)$ and derive the reflexivity of equality from this predicate via the rule

$$x = x \texttt{:-} HU(x).$$

Alternatively we can add explicit assertion $c = c$ for every constant in the logic program.

7.4.4.2. Inequality

If the underlying LP engine does not provide a built-in inequality predicate, we can simulate inequality by introducing a dedicated inequality predicate which has to be axiomatized correctly. This involves to state that inequality is symmetric, antireflexive and that no two constants may be equivalent and inequivalent at the same time:

$$\begin{gathered} Y \neq X \texttt{:-} X \neq Y. \\ \texttt{:-} (X \neq Y), (X = Y). \end{gathered}$$

The later rule requires integrity constraints and captures (through its combination with the reflexivity of equivalence) the anti-reflexivity of inequivalence.

[6] Usage of equality in the head leads to a re-definition of the built-in predicate.

7.4.4.3. Integrity constraints

Integrity constraints can be simulated by introducing a dedicated predicate IC which is used as the head of every integrity constraint. If the knowledge base is inconsistent, i.e. unsatisfiable in the DL sense, the extension of IC is non-empty. We can therefore check for the emptiness of the extension of IC to ensure that the knowledge base is satisfiable.

7.5. Materialization

Incremental maintenance of materializations, as presented in Chapter 6 on page 145, is implemented in the KAON Datalog engine. In case of the materialization of a predicate all changes to facts relevant for the predicate and the rule set defining a predicate are monitored.

The maintenance process is carried out as follows. When a program is designated for materialization, all maintenance rules are generated, the maintenance program itself is evaluated and the extension of all predicates P designated for materialization is stored explicitly. The maintenance program is then used for evaluation instead of the original program which is kept as auxiliary information to track changes to rules. All rules of the original program which define non-materialized predicates are added to the maintenance program.

Updates to facts are then handled in a transactional manner. All individual changes are put into the appropriate p_{edb}^{Ins} and p_{edb}^{Del} predicates. Committing the transaction automatically triggers the evaluation of the maintenance rules. After this evaluation, the extensions of all materialized predicates P are updated by adding the extension of P_{idb}^{Plus} and removing the extension of P_{idb}^{Minus}. Similarly, the extension of all extensional predicates P_{edb} is updated by adding P^{Ins} and removing P^{Del}. As a last step, the extension of P^{Ins} and all other auxiliary predicates are cleared for a new evaluation.

Changes in rules are carried out in the three phase process described in Section 6.4 on page 159: First, the new maintenance rules of rule change are generated. Then, the maintenance program is evaluated and the extensions of materialized predicates are updated as described for the change of facts. As the last step, the maintenance rules for rule change are replaced with maintenance rules for fact changes.

7.6. Related Work

We can find related work in two areas. Firstly, several projects besides KAON attempt to provide general infrastructures for ontology-based applications. Secondly, several APIs for predecessors of OWL have been proposed and are related to the OWL API. The reader may note that a survey of general ontology-based tools such as editors and reasoners are beyond the scope of this section[7].

7.6.1. Infrastructures for ontology-based applications

RDFSuite RDFSuite (Alexaki et al., 2001) is a suite of tools for RDF management. The RDFSuite includes a validating parser for RDF, a RDF Schema specific database (RSSDB) and the RDF Query Language (RQL) – a functional query language for querying RDF repositories. RDFSuite departs from the official RDF semantics and disallows cycles and multiple inheritance in the RDF subsumption hierarchies. RQL queries are rewritten into a set of SQL99 queries, which are executed by the underlying database, which is used for physical storage and query execution.

Sesame Sesame (Broekstra et al., 2002) is an architecture for storing and querying RDF data. It consists of an RDF parser, RQL query module and an administration module for managing RDF models. Sesame fully supports the RDF Schema ontology language. Sesame implements the entailments sanctioned by RDFS at the time data is uploaded to the server and can incrementally maintain this data using a TMS-based approach. Sesame can, however, only store OWL and query ground RDF data. The Sesame system has been successfully deployed as a functional component for RDF support in the KAON SERVER.

Jena Jena (McBride, 2001) is a Java framework for building Semantic Web applications. The latest version of Jena provides a programmatic environment for RDF, RDFS and OWL, including a rule-based inference engine. Jena grew out of work in the HP Labs Semantic Web Programme. The Jena framework includes an RDF API and an OWL API. Jena takes an RDF-centric view, which treats RDF triples as the core of RDF(S) and OWL. Therefore, the Jena OWL API is implemented as a wrapper around the RDF API. Similar to our approach, the Jena APIs recently allow several implementations, which may provide in-memory and persistent storage. Ground RDF data can be queried using the RDF data query language (RDQL)

[7] Cf. (Gómez-Pérez, 2002) for an overview representing the state of the art in 2002.

query language. The inference subsystem is designed to allow a range of inference engines to be used for the purposes of Jena. The connection with the engines is made on the RDF triple level, viz. reasoning with OWL is achieved via an axiomatization of the OWL vocabulary using a single triple predicate. Jena includes a set of predefined reasoners, whose expressivity ranges from reasoning with the transitivity of class and property hierarchies to a generic rule-based reasoner. The later supports user defined rules and employs a hybrid execution strategy, which combines forward chaining based on the RETE algorithm (Forgy, 1982) with tabled backward chaining (Swift & Warren, 1994). Reasoning for RDFS and OWL is achieved by instantiating the generic rule based reasoner with a predefined set of rules. While we share the goal of Jena to provide a general framework for building Semantic Web applications, we do not take the RDF-centric view and consider RDF purely as an exchange syntax. Our implementation of the ontology API completely abstracts from RDF. Similarly, and detailed in the previous part, our strategy of rule-based reasoning with OWL, provides a precise characterization of the incompleteness.

4Suite Server 4Suite Server (4SS)[8] is a platform for XML and RDF processing. Among the provided tools is a RDF data repository. Access to this repository is supported through a dedicated API and a query and inference language called Versa. 4SS provides the data infrastructure of a full database management system, including transactions and concurrency support, access control and a variety of management tools. Versa, however, only provides a specialized language for addressing and querying an RDF model. It allows traversal of arcs, processing of node contents and general expression evaluation, but is not aware of RDF or OWL Semantics.

7.6.2. Ontology and Knowledge Base APIs

There is a long tradition for providing programmatic access to knowledge based systems. However, most of the previous work has been centered around protocols, such as Open Knowledge Base Connectivity (OKBC) (Chaudhri et al., 1998) and Generic Frame Protocol (GFP) (Chaudhri et al., 1997), which are application programming interfaces for accessing knowledge bases stored in knowledge representation systems. Such protocol-centric approaches automatically assume a client-server architecture for application development. However, our approach is rather component-based since our intention is to develop a reusable component for developing OWL-based applications, in style of DOM for XML-based applications.

[8] `http://www.fourthought.com/4SuiteServer/`.

Besides the new version of Jena (McBride, 2001), no APIs for the OWL language exist yet.

DAML+OIL interfaces There have been a number of initiatives to provide application interfaces aimed at precursors of OWL such as DAML+OIL (Horrocks et al., 2001). Jena (McBride, 2001) supplies a DAML+OIL interface that provides convenience wrappers around their RDF interface in order to increase the efficiency of manipulating the DAML+OIL fragments embedded in a particular RDF file. Naturally, this approach gives a rather syntax-centric view of DAML+OIL. Additionally, the implementation is bound to a particular RDF implementation. The DAML API by AT&T government solutions is an additional interface to DAML ontologies. It defines a structural interface for the manipulation of and access to DAML ontologies. It is not bound to a particular syntactic representation such as RDF.

Ontology Editors OilEd (Bechhofer et al., 2001) provided a collection of data structures representing DAML+OIL ontologies. The OilEd data structures suffer in a number of ways. The primary drawback is that the functionality is supplied as implementation *classes* rather than *interfaces*, which binds the client to a particular implementation of the model. In addition, support for tracking and recording changes is minimal. Other ontology editors such as OntoEdit (Sure et al., 2002) and Protege (Noy et al., 2000) also expose their internal APIs to offer access to the underlying data structures but experience similar problems since their design is heavily influenced by the purpose of the editors.

7.7. Conclusion

We have presented the prototypical implementation of the DLP compiler. The compiler makes use of the OWL API to access OWL knowledge bases and is part of the Karlsruhe Ontology and Semantic Web Tool Suite (KAON), which enables KAON to provide DLP ABox reasoning with the KAON Datalog engine.

The DLP compiler itself is still prototypical and was mainly used to validate the ideas underlying DLP and to evaluate the performance of DLP with respect to ABox reasoning. The outcome of this evaluation is presented in the next chapter. The incremental maintenance of a materialization has been implemented as part of the KAON Datalog engine and can now be used in all ontology-based applications that make use of this KAON component.

KAON itself has become one of the currently leading infrastructures for Semantic Web applications and is successfully used around the world in several public and

industrial research projects. Similarly, the OWL API was well-received by the research community and Semantic Web startup companies and has been repeatedly among the top 1000 downloads in the Sourceforge open source project repository.

8. Performance Analysis

"Was in der Antike das Schicksal bewirkte,
leistet heute die Statistik.
Sie vernichtet Helden und Schurken."
Rainer Kohlmayer in (Kohlmayer, 2000)

This chapter analyses the performance of reasoning with DLP knowledge bases. Our analysis consists of four parts:

1. We elicit how typical knowledge bases using Web ontology languages look like. This is achieved through a statistical analysis (Tempich & Volz, 2003) of DAML+OIL ontologies in the DAML.ORG ontology library, which is the largest collection of Web ontologies available today.

2. We compare the performance of solving three standard $\mathcal{QDL}$ reasoning problems between the current tableaux-based generation of DL reasoners and currently available logic databases. We use simple synthetic $\mathcal{L}_0$ knowledge bases to establish a base line comparison.

3. We study the effect of handling knowledge bases which make use of more expressive DLP languages. In particular, we analyze the effect of handling an equality theory and the effect of handling skolem functions on acyclic TBoxes.

4. We analyze the effect of materializing knowledge bases on the performance of solving instance retrieval problems. To this extent, we measure the time needed to setup a materialization and to carry out incremental maintenance in case of changes to facts.

The organization of the chapter follows these four parts: Section 8.1 introduces our approach to performance analysis and discusses how it is related to previous approaches. Section 8.2 on page 196 presents the systems participating in the performance analysis and describes our testing methodology. Section 8.3 on page 198 summarizes our analysis of the DAML.ORG ontology library (Tempich & Volz, 2003), which motivates the structure of the synthetic knowledge bases used for the subsequent performance analyses. Section 8.4 on page 206 summarizes the results

of the comparative analysis establishing a baseline for the performance relations between tableaux-based DL reasoners, such as Racer, and logic databases, such as XSB. Section 8.5 on page 213 summarizes the results of our analysis of instance retrieval performance on knowledge bases which make use of the more expressive DLP language variants. Section 8.6 on page 219 presents the results of our evaluation of the costs related with the incremental maintenance of materialized knowledge bases. Section 8.7 concludes this chapter on page 222 with a summary of our analyses.

8.1. Introduction

Our methodology to analysis might appear to be overly complex at first sight. The organization into several parts, however, allows to factorize the complexity of the analysis and avoids to measure the combination of several influences on performance at the same time. Our strategy therefore allows to identify and measure the different individual influences on the performance which we can observe.

For example, the comparative analysis presented in Section 8.4 on page 206 clearly identifies the relation between the performance of tableaux-based DL reasoners with logic databases. The comparison is achieved by observing the performance of several different reasoning problems with respect to simple knowledge bases.

The results established in this analysis allow us to generalize the behavior of systems for individual reasoning problems when a knowledge base becomes more complicated. We can then focus on those interesting aspects of individual reasoning problems, where a change of performance behavior would be relevant.

Our analysis of the ontology library might appear to be unrelated to the performance analyses themselves. However, an earlier performance analysis (Heinsohn et al., 1994) suggests that the structure of the knowledge base severely affects the runtime performance of DL reasoners. For example, (Heinsohn et al., 1994) report that the runtime of the Loom system (MacGregor, 1991) grows faster than linearly with the number of properties in the TBox. The analysis of the ontology library is therefore carried out in the attempt to identify the average characterization of knowledge bases.

The average characterizations are then used as structural indicators for the formation of synthetic knowledge bases. The individual performance analyses then use these synthetic knowledge bases to measure the time required to solve particular $\mathcal{QDL}$ reasoning problems.

The usage of synthetic knowledge bases has several benefits: Firstly, knowledge bases can be scaled in size while maintaining the same structural properties. Sec-

ondly, analyses can be carried out in a controlled and repeated manner, which is the most important requirement for any scientific experiment. Thirdly, the homogeneous structure facilitates the empirical assessment of the correctness of provided answers.

Naturally, a performance analysis can be performed using a real knowledge base that is used on a particular reasoner. For example, (Horrocks, 1997) uses the GALEN knowledge base to assess the performance of the FaCT (Horrocks, 1998) reasoner. However, real knowledge bases, in particular Web ontologies, evolve and it is often impossible to apply them repeatedly. Similarly, as our analysis of the library will show, individual ontologies rarely correspond to the average case. Therefore the results of an analysis based on real knowledge bases are hardly representative and the generalization of the results is problematic.

The reader may note that our approach to analysis is also substantially different from the largest previous performance analyses carried out in the Description Logic community (Horrocks & Patel-Schneider, 1998). This previous approach was mainly focused on testing the performance of satisfiability checking and used a sequence of problems with (exponentially) increasing difficulty (Balsiger & Heuerding, 1998; Heuerding & Schwendimann, 1996; Horrocks & Patel-Schneider, 1998). While such problem definitions are effective for testing reasoners and individual algorithms, they are hardly representative and it is

> "... questionable whether performance for worst case examples gives us the right idea of how systems will behave in applications." (Heinsohn et al., 1994).

Limitations Due to the absence of available ABoxes used in conjunction with expressive ontologies in the ontology library, our approach to synthetic data generation for ABoxes is deemed to be subjective and speculative about a possible ABox structure. We make the implicit assumption that properties are used similarly to assoctiations and attributes of classes in object-oriented systems. This behaviour can be observed on the few large ABoxes that are currently available[1]. Our analysis does not encompass datatype properties, datatype property restrictions and instances of those properties.

[1] In particular, TAP `http://tap.semanticweb.org/` and AKTive Space `http://www.aktors.org/`.

8.2. Analyis Environment

This section presents the systems participating in the performance analyses, the host environment used in carrying out the analyses and our testing methodology.

8.2.1. Selection of Systems

The selection of systems used for our analysis is motivated by the following criteria:

1. Active Maintenance
2. Free and public availability
3. State of the art
4. Feature support

To support the comparative analysis, we chose systems from the logic database category and the DL reasoner category.

Logic Databases Our choice fell on the KAON Datalog engine and the XSB system (Sagonas et al., 1994). Other logic databases such as Coral (Ramakrishnan et al., 1994) are no longer actively maintained. Systems like Ontobroker (Decker et al., 1999) were available to the author, but are not freely available to the public due to their commercial status. Therefore, the results of a performance analysis could not be repeated by third parties.

DL Reasoners Our choice fell on the Racer system (Haarslev & Moller, 2001), which is the only DL engine that uses state of the art (tableaux) reasoning algorithms and features support for ABox reasoning. We therefore did not regard Loom (MacGregor, 1991), another publicly available DL reasoner, which uses (incomplete) structural subsumption algorithms. NeoClassic (Patel-Schneider et al., 1991) also uses structural algorithms and appears not to be actively maintained.

8.2.2. System Descriptions

8.2.2.1. KAON Datalog

We used a fresh compilation of the Java-based KAON Datalog engine as it could be obtained from KAON CVS [2] on December 14th, 2003. The KAON Datalog en-

[2] `http://sourceforge.net/projects/KAON/`.

gine allows to use multiple extensional databases with the same program. This feature was used for the $\mathcal{QDL}$ subsumption test, where we simply created a new extensional database that only contained the prototypical instance required for the test and used in conjunction with the program. Since the extensional database did not contain any other instances, we expect limited binding passing possibilities and carried out the subsumption test without applying the Magic Set transformation. For instance retrieval, however, the Magic Set transformation was applied for query answering. The KAON Datalog engine does not provide a built-in support for equality and integrity constraints. We therefore apply the necessary simulations reported in Section 7.4.4 on page 185. In the following, we will refer to the KAON Datalog engine as KAON.

8.2.2.2. XSB

We used the XSB Windows executable (Version 2.6) as obtainable from the XSB Web site. The communication between XSB and the DLP prototype was carried out by embedding XSB into the Java process using the Java Win32Process infrastructure. A new XSB process was created for every test. For the DLP translation of knowledge bases into XSB programs, we used dynamic XSB programs. Dynamic programs contain dynamic predicates which can be changed at runtime (by asserting/retracting rules or facts). The ability to assert and retract facts is required for the $\mathcal{QDL}$ subsumption problem. XSB programs are typically not dynamic. In this case, XSB can only load a given file and issue queries on the extensions of predicates in the program but not manipulate the program and its extensions during runtime. The use of static programs allows XSB to apply optimizations, which would lead to different response times than the ones measured by us. All predicates were tabled. Since XSB does not provide integrity constraints and the built-in equality predicate of XSB can not be used[3], we apply the simulations reported in Section 7.4.4 on page 185.

8.2.2.3. Racer

The version of Racer used in the analysis is the stand-alone Windows Racer server (Version 1.7.12) as obtained from the Racer web site. The communication of Racer was carried out using the TCP/IP protocol. The Racer server was started locally and TCP/IP communication did not involve any network latencies, since the computer was not connected to any computer network during the analysis. The same

[3]Since XSB understands the usage of a built-in predicate in the head of rules as a redefinition of predicates, i.e. loses the predefined axiomatization of the predicate.

Racer server was reused across all tests of an analysis. Before every test, all available knowledge bases in the server were unloaded. In order to facilitate the construction of internal optimizations, the initial loading of a knowledge base was augmented with a test on the consistency of the TBox and the ABox before subsumption and instance / property filler retrieval problems were stated. The reader may note that the support of Racer for ABoxes is incomplete. For example, Racer does not incorporate an equality theory and can therefore not identify individuals as members of a class, if they are equivalent to instances of the class.

8.2.2.4. Host Environment

The analysis was carried out on a IBM T30 Laptop, equipped with a Pentium IV Mobile processor running at 2 GhZ and 512 MB main memory under the Windows XP Professional operating system. Windows XP internally used a 1 GB page file for virtual memory. The tests itself were written in Java and executed using the Java JDK 1.4.1_01.

8.2.3. Testing Methodology

Every test was carried out 5 times on all three systems. All systems were using the same (synthetically generated) knowledge base across the five tests. However, different classes and properties were selected for every repetition of a test. The results stated here represent the average between all 5 invocations of a test. Each test is given a time frame of 30 minutes (1800000 ms) and was aborted if the respective system was unable to solve the problem in time.

Our primary metric is response time, i.e. the time (in milliseconds) that is needed to solve the reasoning problem. This means that we ignore the utilization of system resources, which could be another interesting measure for performance. However, we assume that most Web ontology reasoners are going to be servers, for which it is nowadays cheap to buy the necessary memory and CPU to ensure that these factors are not the bottleneck for performance.

8.3. Ontology Library Analysis

We provided the first statistical analysis of the DAML.org library of ontologies[4] in (Tempich & Volz, 2003). Our choice to analyze this collection of ontologies is motivated by its sheer size. It is currently the largest collection of available ontologies.

[4] Available at `http://www.daml.org/ontologies`.

Secondary reasons for our choice are that we are not related with the authors of the ontologies in any form[5]. Additionally, most ontologies are created by different people with diverse technical backgrounds (ranging from students to researchers). In this sense, the user group can be understood as being representative for the current state of the Semantic Web and the ontologies are mostly not created by expert logicians.

We present a brief summary of our analysis (Tempich & Volz, 2003) in the following. The individual results of the analysis can be found in Appendix D on page 253. We focus on the aspects that are relevant for the synthesis of knowledge bases used in the performance analyses. Further aspects, e.g. the application of clustering to categorize the individual ontologies, are discussed in (Tempich & Volz, 2003).

8.3.1. Library Content

The analysis was carried out prior to the analysis of the ontologies presented in Section 5.3.2 on page 128. At time of the analysis (July 2003), the DAML.ORG library contained 280 ontologies. Since only very few OWL ontologies were available at the time and the correctness of converters from DAML+OIL to OWL format could not be verified empirically, we decided to analyze ontologies in the DAML+OIL format, which roughly correspond to OWL DL.

Unfortunately, 50% of the ontologies contained fundamental errors[6] and could not be processed. In order to avoid a manipulation of the ontologies, we did not make any attempt to fix these problems. Therefore, we could only parse 140 ontologies. The correctness of RDF is, however, not the only relevant property that an ontology must meet in order to be processable. For example, it has to use the DAML+OIL/OWL vocabulary correctly. Due to further errors like the usage of wrong vocabulary, another 45 of the parse-able RDF documents could not be processed as DAML+OIL ontologies. This suggests that we need further heuristics to make use of these ontologies in reasoners, i.e. deriving the correct vocabulary for the DAML+OIL language. We did not make use of those heuristics to have an exact representation of the definitions made in ontologies. In consequence, only 95 ontologies could be used for our analysis.

This has serious statistical consequences on the representativeness of the results of the analysis. Due to the small size of the collection, we have to understand the averages established here as rough indicators instead as "hard" scientific facts. The later can only be established on a larger collection of ontologies. The methodology

[5] Viz. the selection is not based with the aim to facilitate a particular outcome of the analysis.

[6] Ranging from syntactic XML and RDF errors to HTTP errors indicating that the given ontology cannot be found.

lain out here may provide future help to reassess our results when the number of Web ontologies grows.

The analysis itself is based on the structural properties of asserted information, hence no reasoning was performed. This has various consequences, e.g. (due to the semantics of Description Logics) every class description actually has at least one entailed superclass, namely $\top$. Since we measured the asserted super-classes only, it is possible that class descriptions have no super-classes.

The analysis was carried out using a simple Java class that counted the elements of ontologies. The correctness of the numbers additionally depends on the correctness of the DAML+OIL library of Jena Version 1.6.1 which was used to represent the ontologies programmatically.

Constructor	Av.	Std. Dev.	Median	Min.	Max.	C.O.V.[7]
Class Descriptions	175,2	1016,39	19	1	9795	5,8
- Named Classes	166,89	1018,21	5	1	9795	6,1
- Restrictions	19,13	44,52	7	0	327	2,33
– Universal Restrictions	13,29	37,7	4	0	327	2,84
– Existential Restrictions	1,97	15,33	0	0	146	7,79
– Number Restrictions	8,96	33,95	1	0	326	3,79
— Number Restr. (Lite)	8,86	33,93	1	0	326	3,83
- Enumeration	0,33	1,72	0	0	16	5,26
- Boolean Constructors	1,45	8,62	0	0	78	5,94

Table 8.1.: Average Usage of Class Constructors Across Ontologies

8.3.2. Average Characterizations

8.3.2.1. Absolute Numbers

The first part of our analysis was concerned with simply counting the usage of certain class constructors, in order to estimate probabilities that measure the usage frequency of each constructor. Table 8.1 summarizes the average usage of the individual class constructors in the ontologies. One important aspect of the summary given in Table 8.1 is that the standard deviation is very high and that the median and the average are often very different. These facts show that the particular ontologies vary tremendously in their usage of individual class constructors, i.e. the distributions are highly skewed. In such cases, the median is usually considered to

[7] The coefficient of variation (C.O.V.) measures the *relative* dispersion and is defined as $\frac{\text{Std. Dev.}}{\text{Average}}$.

be a statistically more representative characterization of the average case than the average itself.

As we can see, atomic named classes and various types of restrictions are the predominant form of class descriptions. Even though the average of classes is higher than the average of restrictions, an ontology typically contains more restrictions than atomic class names (as indicated by the median).

One interesting aspect for our context is that only two ontologies actually used number restrictions that could not be expressed in $\mathcal{L}_1$. Additionally we can see that enumerations are rarely used, even considering the hasValue property restriction. Boolean constructors are also used rarely: only ten percent of the ontologies used negation, intersection or union as class constructors.

Property	Absolute Numbers					
	Average	Std. Dev.	Median	Min.	Max.	C.O.V.
Properties	28,41	43,47	13	0	269	1,53
Untyped	7,51	23,34	0	0	176	3,11
Datatype	12,57	23,15	4	0	145	1,84
Object	8,34	31,26	2	0	269	3,75
Transitive	0,22	0,66	0	0	3	2,96
Functional	0,56	2,21	0	0	14	3,95
InverseFunctional	0,02	0,14	0	0	1	6,86
Inverse	0,59	3,62	0	0	34	6,14
Domain	0,25	0,39	0	0	1,23	1,59
Range	0,26	0,41	0	0	1,22	1,57

Table 8.2.: Average Usage of Property Definitions

Table 8.2 presents the absolute counts of property assertions. Since we were dealing with DAML+OIL, properties do not necessarily have to be typed. The predominant form of properties were datatype properties. We can also see from Table 8.2 that only few property characteristics are specified. In particular, inverse functionality was used rarely. Interestingly, only every fourth property has asserted global domain and ranges specifications.

8.3.2.2. Relative Numbers

The second part of our analysis is concerned with the ratio of different primitives in ontologies. The variability of these ratios is smaller than the average counts (cf. Table 8.3) since we aggregated relativized numbers.

Ratios	Average	Std Dev	Median	Min	Max	C.O.V.
Prim. Class /Constructor	50%	0,34	39%	6%	100%	0,67
Obj. Prop. / Prop.	24%	0,27	16%	0%	100%	1,12
Dat. Prop. / Prop .	52%	0,38	67%	0%	100%	0,72
Trans. Prop. / Obj. Prop.	3%	0,08	0	0	0,4	3,13
Obj. Prop./ Prim. Class	0,61	0,83	0,50	-	5,57	1,35
Dat. Prop./ Prim. Class	2,25	3,33	1,00	-	15,75	1,48
Prop. / Prim. Class	3,54	3,89	2,00	-	21,00	1,10
Ex. Rest. / Rest.	1%	0,08	0%	0%	65%	5,79
Univ. Rest. / Rest.	48%	0,44	52%	0%	100%	0,91
Card. Rest./ Rest.	34%	0,38	20%	0%	100%	1,11
Rest./Primitive	2,32	2,70	1,50	-	16,00	1,17
Asserted Ind. / Primitive	0,60	4,18	-	-	40,50	6,97

Table 8.3.: Ratio between Selected Language Primitives (Across all Ontologies)

One aspect that can be observed is that the ratio between properties and named classes is very low, since the median of properties per class is two. The reader may recall that DAML+OIL allows for untyped properties, for which the specification of being datatype or object property is missing. We will therefore assume that half of the properties are actually object properties. As a result of this assumption, we will use one object property per named class as the representative case in the synthetic ontologies.

The low average of properties per class description can be interpreted as being one of the major differences between typical TBoxes and object database schemas, where this ratio would be higher. Interestingly, datatype properties are the predominant type of properties. It is important to note that some of the ontologies, particularly those with high numbers of classes, do not contain any properties at all. Only some 3% of the defined object properties were declared transitive. We will therefore ignore property transitivity in the generated synthetic knowledge bases.

Among the restrictions, universal restrictions are predominant, i.e. almost half of all restrictions are universal restrictions. Typically, a primitive class is defined using 2,32 restrictions (median 1,5).

8.3.3. Distributions in Class Definitions

The third part of analysis is concerned with determining frequency distributions of the constructors contained in class axioms. Since most of the constructors do not occur in significant number, we concentrated on two distributions:

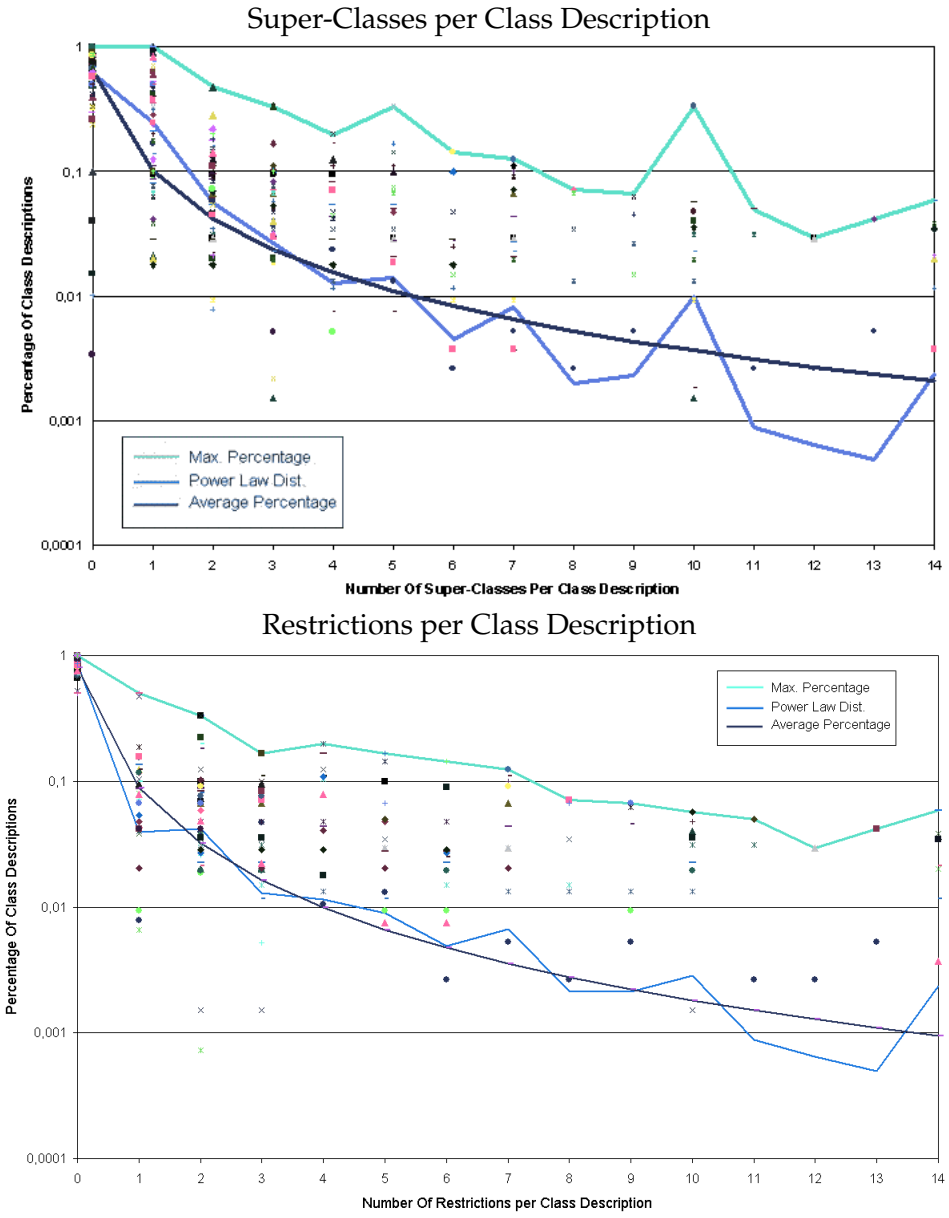


Figure 8.1.: Distributions in Class Descriptions

- Number of assigned named super-classes per class definition;
- Number of restrictions used in the definition of class definitions.

8.3.3.1. Named Super-Classes and Restrictions

The Figure 8.1 depicts the frequency distribution of super classes and restrictions per class description. In these figures, the actual number of super-classes (restrictions) constitutes the x-axis of the graph.[8] The y-axis, which is drawn at logarithmic scale, represents the percentage of classes in an ontology for which the respective number of super-classes (restrictions) can be observed. Every such observation in the individual ontologies is depicted by a data point. Additionally, the figure shows three lines. These lines depict the observed average and maximum percentage across all ontologies as well as an approximation of the average percentage via a standard distribution.

It is important to recall in these figures that our analysis was performed on the syntactic declarations. Therefore, we do not consider that we can infer for every class that it has at least one subclass ($\bot$) and one superclass ($\top$). We actually found ontologies that were slightly inconsistent in this respect. Several ontologies redeclared the symbol for $\top$, i.e. DAML:THING, in another namespace (usually the namespace of the ontology). Similarly, $\top$ was sometimes explicitly assigned as super-class. These situations were, however, not treated separately in the analysis, since we intended not to manipulate the ontologies.

8.3.3.2. Approximating Distributions

In order to simulate the distributions, we need to approximate the observed distributions with a standard distribution. Our empirical tests soon led to the choice of Power Law distributions, which appear to fit the observed averages very well. Power Law distributions can be observed in many real-life situations, such as linkage between web sites or human urban habitation. Power Law distributions suggest in our setting that there is a big percentage of classes that have only few superclasses (restrictions) and a small percentage of class descriptions that have many super-classes (restrictions). The Power Law distribution is defined as follows:

$$P(r) = c * r^{-a}$$

[8] The last value (14) aggregates all greater numbers, hence the percentage of classes that feature a greater number of elements is also aggregated.

P(r) measures the size of event r, where r is the rank in descending order on size a of the distribution and c is a scaling constant.

In order to determine the parameters c and a, we fitted the distribution to the observed average numbers by performing a linear regression of $log(P(r))$ on $log(r)$ where the parameters $log(c)$ is the slope and a is the intercept of the curve $log(P(r)) = log(c) + a * log(r)$.

Using the linear regression, we can determine[9] the scaling parameter $c = 0.2$ and $a = 1.7$ for the super-class distribution and $c = 0.2$ and $a = 2.0$ for the restriction distribution. The χ^2 test shows for both approximations that they are good hypotheses for the observed average distributions. Similarly, the standard deviation of observed values from the approximation only differs 7% (30%) in the case of restrictions (super-classes) from the deviation of observed values with the observed average.

8.3.4. Discussion

Our analysis shows that individual ontologies vary tremendously not only in size but also in their average use of ontological constructors. This variation leads to high standard deviations and makes it difficult to identify the average case. Nevertheless, the analysis helps to assess previous approaches for synthetic generation of knowledge bases. While (Elhaik et al., 1998) develop a very sophisticated approach for the generation of knowledge bases, the approach appears to be too complicated in practise. This is due to the fact that it requires distributions for the occurrence of class constructors at certain depths in class descriptions. These distributions can, due to the small number of available ontologies and their high variance, not be realistically provided. Similarly, the analysis shows that the more simplistic approach of (Heinsohn et al., 1994), which assume that 80% of the classes are primitive and each class definition contains one or two super-classes, zero, one or two value restrictions and zero, one or two number restrictions, is no longer valid. The analysis of the distribution of super-classes clearly shows, that the Power Law distribution fits the actual distributions for super-classes and restrictions much better.

In the following, we do not instantiate the Power Law distributions to simplify our empirical analysis of the correctness of the obtained answers. The knowledge base generation therefore follows (in style) (Heinsohn et al., 1994), with the addition that the number of subclasses and restrictions per named class are adapted to reflect the outcome of our analysis.

[9] For the purpose of presentation, we round the actual numbers to the 1/10 digit.

8.4. Comparative Performance Analysis

This section presents the first set of results of our analysis of the performance of DLP.

8.4.1. Goals and Assumptions

The aim of the comparative performance analysis is, as mentioned above, to compare the basic performance of the two logic databases and the Racer Description Logic reasoner with respect to the time needed to solve the following three basic $\mathcal{QDL}$ reasoning problems:

1. Retrieval of class instances for a random class;
2. Retrieval of property fillers for a random property;
3. Class subsumption between two random classes.

Symbol	Variant	Tax. Depth	No. Classes
TS	Small	3	40
TM	Medium	5	364
TL	Large	7	3280

Table 8.4.: TBox Variants

8.4.2. Knowledge Bases

The $\mathcal{QDL}$ reasoning problems were stated against various synthetically generated knowledge bases (KB). The TBox of each KB is varied in its taxonomic depth (cf. Table 8.4 above). Each TBox was constituted by a symmetric taxonomy of primitive classes, i.e. only contained inclusion axioms. Each (non-leaf) class has exactly three subclasses. The ABox of each KB is generated by varying the number of individuals per class (cf. Table 8.5 below).

We consider the statement that the performance of DL systems is dependent on the number of properties present in the TBox (Heinsohn et al., 1994) and considered three different variants of generated properties and property fillers in the KB (cf.

[10] We do not assert members of $\top$.

Symbol	Variant	No. Individuals / Class[10]
IS	Small	3
IM	Medium	9
IL	Large	15

Table 8.5.: ABox Variants

Table 8.6). The first variant (P0) of TBoxes did not contain any properties. This variant approximates the taxonomies of thesauri such as WordNet or UNSPC, which currently constitute a large portion of Web ontologies that are in use. Our goal with this test is to see how the very basic task of taking the taxonomy into account when retrieving the extension of a class, is improved. The taxonomy is constituted by a symmetric tree of classes.

The second variant (P1) of TBoxes contains exactly one object property per class[11] and fills this property for every third individual (connecting the individual to the previously generated individual). The third variant (PF) of TBoxes contained exactly 200 object properties and instantiated a single one of these properties (with uniform distribution) on every individual.

Symbol	Variant	No. Properties	No. Prop. Fillers / Class
P0	No Properties	0	0
P1	One Prop. / Class	No. Classes	1/3
PF	Fixed No. of Prop.	200	No. of Individuals

Table 8.6.: Property Variants

We generated a total number of 27 knowledge bases by instantiating all possible parameter combinations (cf. Table 8.7 for the number of classes, properties and property fillers in every KB).[12]

[11] The reader may recall that DAML+OIL allows for untyped properties, for which the specification of being datatype or object property is missing. We will therefore assume that half of the observed properties are actually object properties. Due to the observed dispersion.

[12] Since each of three above mentioned reasoning problems were instantiated on every knowledge base, 81 different tests were carried out. Every test was repeated 5 times, leading to 405 individual tests. The time out of 30 minutes per test suggests a maximum running time of slightly more than one week.

[13] Since we do not assert members of $\top$ the number of individuals in the knowledge base are calculated by (# Classes - 1) * # Individuals/Class.

Variant			Classes	Individuals[13]	Properties	Prop. Fillers
P0	TS	IS	40	117	0	0
P0	TS	IM	40	351	0	0
P0	TS	IL	40	585	0	0
P0	TM	IS	364	1089	0	0
P0	TM	IM	364	3267	0	0
P0	TM	IL	364	5445	0	0
P0	TL	IS	3280	9837	0	0
P0	TL	IM	3280	29511	0	0
P0	TL	IL	3280	49185	0	0
PF	TS	IS	40	117	200	117
PF	TS	IM	40	351	200	351
PF	TS	IL	40	585	200	585
PF	TM	IS	364	1089	200	1089
PF	TM	IM	364	3267	200	3267
PF	TM	IL	364	5445	200	5445
PF	TL	IS	3280	9837	200	9837
PF	TL	IM	3280	29511	200	29511
PF	TL	IL	3280	49185	200	49185
P1	TS	IS	40	117	40	40
P1	TS	IM	40	351	40	120
P1	TS	IL	40	585	40	200
P1	TM	IS	364	1089	364	364
P1	TM	IM	364	3267	364	1092
P1	TM	IL	364	5445	364	1820
P1	TL	IS	3280	9837	3280	3280
P1	TL	IM	3280	29511	3280	9840
P1	TL	IL	3280	49185	3280	16400

Table 8.7.: Input Knowledge Base Size

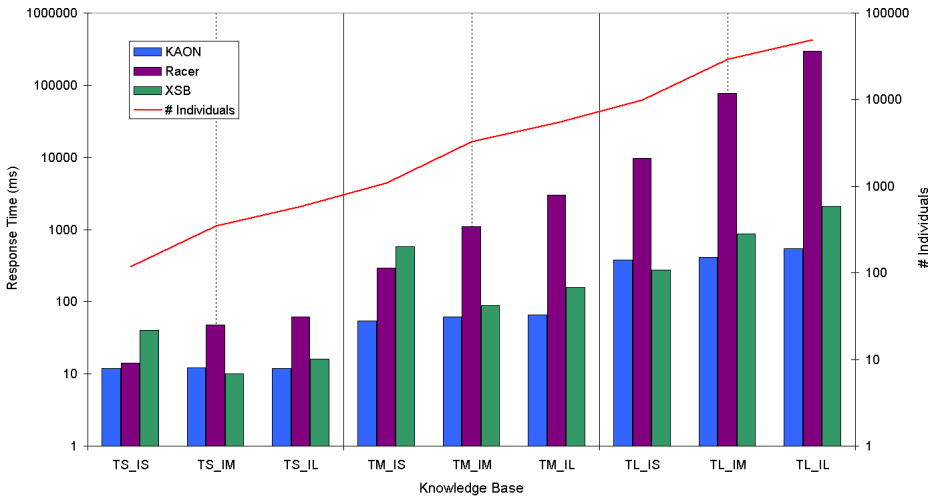


Figure 8.2.: Instance Retrieval on PO KBs

8.4.3. Results

The individual results of the tests can be found in Appendix B.1.1 on page 237. As before, we will summarize the results here and pick interesting data for the purposes of this presentation.

8.4.3.1. Instance Retrieval

P0 Knowledge Bases We first consider the results of the instance retrieval tests on the $P0$ family of knowledge bases, which did not contain any properties. Figure 8.2 describes the results of these tests, where the average response time for each query is drawn on a logarithmic scale. The worst average response time by all systems was given for the largest knowledge base (TL IL).

KAON shows the best performance for instance retrieval across all knowledge bases. The worst average response time of KAON was 0,5 seconds. XSB was in average 2,65 times slower than KAON, while its' worst response time was 2 seconds (4 times slower than KAON). The reader may note that, if instance retrieval was the only query to be answered by XSB, the use of compiled XSB programs empirically shows similar performance of XSB in comparison to KAON. The performance of both systems shows close to linear growth with respect to the number of instances.

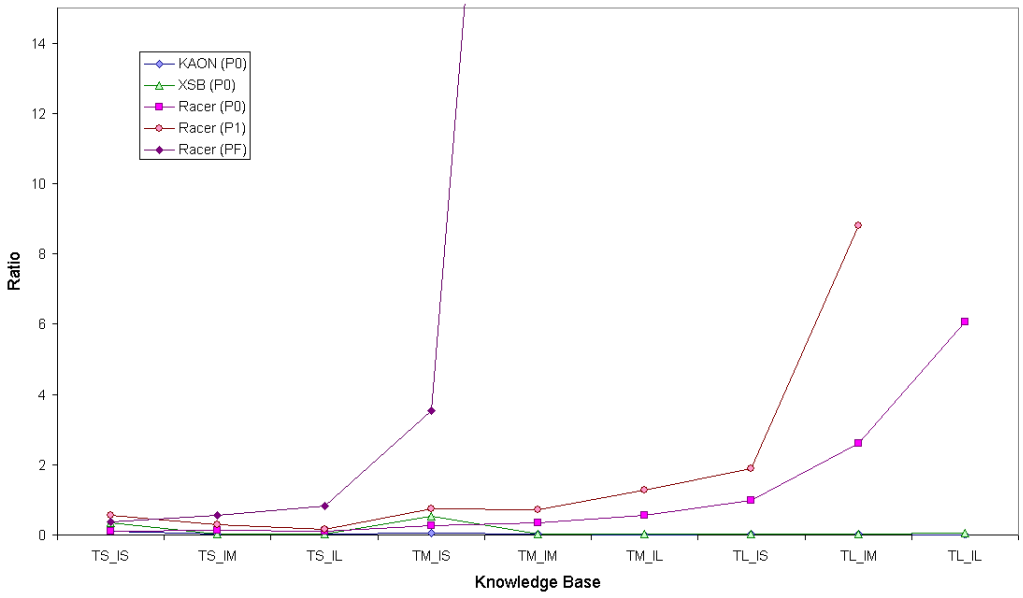


Figure 8.3.: Ratio of Response Time to Number of Individuals

The performance of Racer was highly dependent on the number of individuals and grows exponentially with the number of individuals (cf. Figure 8.3). The response time was still acceptable for P0 knowledge bases. Even on the largest P0 knowledge base, which included almost 50000 individuals, Racer was able to solve the reasoning problem within 5 minutes.

P1 Knowledge Bases The presence of properties and property fillers did almost not affect the performance of the two logic databases. However, the exponential effect on the performance of Racer gained tremendous momentum (cf. line P1 in Figure 8.3), leading to the situation that Racer was unable to answer any instance retrieval problem posed on the largest P1 knowledge base within the given time frame of 30 minutes per test.

PF Knowledge Bases In order to investigate whether the increased time needed by Racer is due to the presence of properties in the TBox such as suggested by (Heinsohn et al., 1994), we investigated whether a fixed (and small) number of 200 properties would make a difference. However, we discovered that the number of properties actually does not matter. Surprisingly, the strongest influence on the performance of Racer with respect to instance retrieval appears to be the number of property fillers. In fact, Racer was not able to solve an instance retrieval problem within the given time frame of 30 minutes[14] when more than 10000 property fillers were present. The performance of XSB and KAON, on the other hand, was not noticeably affected by the presence of a large number of property fillers.

Summary We can summarize that logic databases outperform the current version of Racer by order of several magnitudes. The exponential dependency of Racer to the size of the knowledge base prevents its practical use for instance retrieval on large knowledge bases in particular when property fillers are present.

Logic databases, however, expose satisfactory performance. None of the queries took KAON (XSB) more than 1 second (4.5 seconds) to answer. The analysis therefore suggests that logic databases show sufficient performance for real-life knowledge bases, which are similar in size to the evaluated synthetic knowledge bases.

8.4.3.2. Property Filler Retrieval

The retrieval of property fillers from our knowledge base corresponds to a simple lookup of explicit assertions, since no entailments such as sanctioned by a property hierarchy are necessary.

Figure 8.4[15] summarizes the results of the individual property filler tests. Clearly, the retrieval of explicit information in logic databases is trivial. Due to this fact, we

[14] In order to test whether Racer was slightly slower than 30 minutes, we did not stop one test. The test was running for 8 hours before it was eventually aborted. Similarly, the otherwise moderate memory consumption of Racer was not present anymore.

[15] The different knowledge bases are sorted by the number of property fillers.

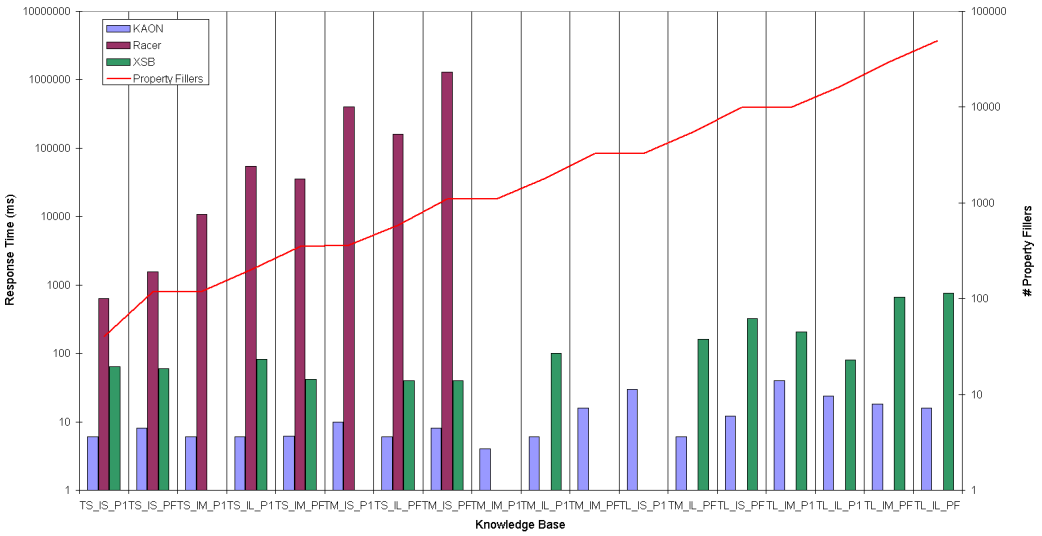


Figure 8.4.: Response Time for Property Retrieval Tests

were sometimes unable to measure the response time of XSB. In average, KAON (XSB) was able to solve the posed property retrieval problems in 13 (146) ms.

Interestingly, Racer performs even worse for the retrieval of property fillers than for instance retrieval. Racer was typically unable to solve all property filler problems in the given time limit, when more than 1000 property fillers were present. If more than 200 property fillers were present, we could only measure response times above 1 minute.

The results for property fillers underline our conclusion for instance retrieval. Apparently, Racer is only of limited use, even for the smallest knowledge bases evaluated here.

8.4.3.3. Class Subsumption

The results of the class subsumption tests are summarized in Table 8.8. As expected, the two decades of research in DL systems, which was almost entirely focused to optimize this type of reasoning problem, show their effect. Class subsumption problems are the stronghold of Racer. On average, Racer solves the posed subsumption problems within 30 ms[16].

The largest TBoxes considered in our analysis are primitive and of minuscule size

[16]The median response time, however, is 0 ms.

KB			KAON (ms)	Racer (ms)	XSB (ms)
P0	TS	IS	22	2	0
PF	TS	IS	38,2	0	180,2
P1	TS	IS	22,2	16	62
P0	TS	IM	14	2	20
PF	TS	IM	52	0	220,4
P1	TS	IM	22	0	0
P0	TS	IL	18	0	0
PF	TS	IL	50	0	200
P1	TS	IL	22	2	240,6
P0	TM	IS	98	4	300,2
PF	TM	IS	118,4	4	120,2
P1	TM	IS	146,2	6	0
P0	TM	IM	90,2	2	0
PF	TM	IM	110,2	0	60
P1	TM	IM	122	0	0
P0	TM	IL	74	2	0
PF	TM	IL	88	114,2	200,4
P1	TM	IL	146,4	4	200,6
P0	TL	IS	767,2	2	0
PF	TL	IS	729	6	280,6
P1	TL	IS	1466,2	0	0
P0	TL	IM	638,8	6	0
PF	TL	IM	691	6,2	300,6
P1	TL	IM	1704,4	86,2	262,6
P0	TL	IL	707	296,4	24
PF	TL	IL	570,8	10	280,2
P1	TL	IL	1347,8	292,4	200,2

Table 8.8.: Response Time to Class Subsumption Problems

compared to other knowledge bases where Racer has been reported (Haarslev & Moeller, 2001) to solve classification-related problems in a satisfactory manner.

The logic databases, however, also performed well in our analysis. Notably, KAON (XSB) was in average 11,4 (3,7) times slower than Racer. XSB outperformed KAON and took at most 0,3 seconds to answer the posed subsumption problem. Similarly, KAON needed at most 1,7 seconds to answer a class subsumption problem. Interestingly, both worst case response times were not observed on the largest knowledge bases. Table 8.8 additionally suggests that the response time of KAON depends on the size of the knowledge base. This conjecture cannot be repeated for XSB whose performance variations appear to be random.

The results for logic databases should, however, not be understood as an indication that logic databases are well-suited as DL classifiers. The asymmetry of DLP with

respect to class subsumption described in Section 5.4.4.1 on page 135, i.e. we can only ask whether a $\mathcal{L}_i^L$ class subsumes a $\mathcal{L}_i^R$ class, cannot be overcome.

8.4.4. Discussion

Our results show that the performance of Racer in all ABox-related problems is significantly worse than the performance of logic databases. This emphasizes the contribution of this thesis, which enables ABox reasoning for almost all present Web ontologies in practise.

The bad performance of Racer is mainly due to the fact that the tableau reasoning procedure employed within Racer provides a proof or a refutation for one class-instance pair (Haarslev & Moeller, 2000), whereas the SLG-WAM resolution employed in XSB or Magic Sets work with sets of instances.

Further, one may observe that the performance of Racer deteriorates with the presence of property instances. The reason for the bad performance of Racer clearly follows from the current implementation in Racer[17], which translates property filler retrieval on a property R to consistency checking of an ABox $\mathcal{A}$[18]. This procedure requires |# Individuals in $\mathcal{A}|^2$ ABox consistency checks. Therefore, Racer has to perform more than $2,4 * 10^9$ ABox consistency checks for the largest knowledge base (TL IL).

What also has become clear from this analysis, is that logic databases can be practically used for DL subsumption tests, if the users are interested whether a $\mathcal{L}_i^L$ class subsumes a $\mathcal{L}_i^R$ class. Generally, however, we conjecture that a dual reasoning strategy employing logic databases and DL classifiers in parallel is needed to fully cover all aspects of Web ontology reasoning.

8.5. Expressivity Performance Analysis

This section presents the second part of analysis of DLP reasoning performance. We rely on the insights obtained in the comparative analysis and concentrate on the effects of handling the more expressive language constructs such as found in $\mathcal{L}_{i>0}$ on the instance retrieval problem.

[17] Personal communication with an author of the Racer system.

[18] For all pairs (i, j) of individuals in an ABox $\mathcal{A}$, (i, j) is a filler of the property R if the ABox $\mathcal{A} \cup \{\forall R.X(i), (\neg X)(j)\}$ is inconsistent.

8.5.1. Goals and Assumptions

8.5.1.1. Goals

We particularly study the effect of incorporating an equality theory into the logic program as needed for handling the constructs of $\mathcal{L}_1$. Similarly, we study the effects of evaluating logic programs with skolem functions as needed for $\mathcal{L}_3$.

8.5.1.2. Assumptions

We do not analyze the effects of handling $\mathcal{L}_2$ class constructors, since the instantiation of these constructors are transformed into integrity constraints, which have to be checked after every update to the knowledge base and initially when the KB is loaded. They therefore do not affect the performance of instance retrieval, which is the focus of this section. Moreover, both $\bot$ and the class complement constructor $\neg C$ were only used very rarely in the library.

Since the added class constructors do not concern property hierarchies, the analysis does not measure the performance of retrieving property fillers, where the impact of equality and skolem functions should be similar to instance retrieval.

The superior performance of Racer with respect to subsumption has already been shown. Hence, we do not measure the performance of classification. This is also due to the fact that we can only slightly reduce the above mentioned asymmetry between $\mathcal{L}^R$ and $\mathcal{L}^L$ in $\mathcal{L}_3$ by the existential restriction constructor.

8.5.1.3. A-priori Considerations

For the purpose of this analysis, we had to alter the analysis environment (cf. Section 8.2 on page 196) slightly. Firstly, we did not consider Racer, since Racer does not support the equality of individuals. Secondly, we generated static XSB programs since the analysis of dynamic XSB programs generally led to a time out on all tests. The reader may note that the necessity of this change suggests that XSB can no longer be used to solve the class subsumption problem for $\mathcal{L}_{i\geq 1}$ knowledge bases, since the simulation would then require to generate and analyse the logic program for every subsumption test. However, since the previous analysis suggests to use DL reasoners for subsumption testing, we decided to generate static XSB programs to obtain at least some results.

The fact that KAON was able to answer (some) instance retrieval queries in time suggests that the binding passing strategy of Magic Sets is particularly important for equality. This can be accounted to the fact that, due to the axiom of reflexivity,

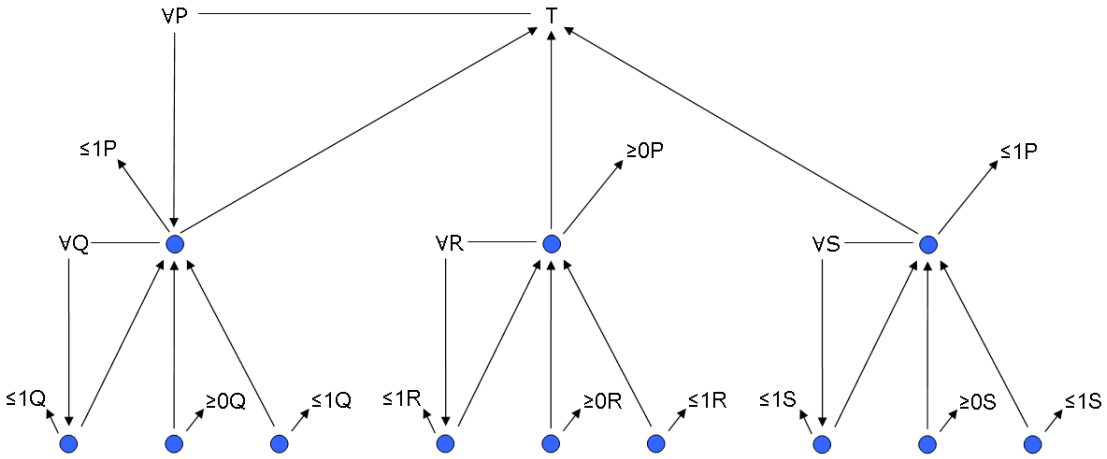


Figure 8.5.: KB structure

every element of the Herbrand universe occurs in at least one implicit equality assertion. Magic Sets allow to choose the subset of equality assertions which are relevant for the query.

We can also conjecture that a pure top-down evaluation such as the plain SLDNF-resolution which is applied in many Prolog systems will not be successful for the analysis of languages containing equality, since the axioms of symmetry and transitivity will lead to infinite recursion. Tabling strategies such as applied in XSB generally make sense as well, since they avoid unnecessary re-derivations.

8.5.2. Knowledge Bases with Equality

We analyze the performance of $\mathcal{L}_1$ knowledge bases by extending the P0 TBox variant of the symmetric class tree used in the previous analysis. The extension simulates the median observation in the ontology library and introduces two restrictions for every non-leaf, non-root and named class of the symmetric class tree. We introduce one restriction for the root class of the tree and for every named leaf class. Thereby 4/3 object property restrictions are created per class[19], which is half of the average number of restrictions per class observed in the analysis of the DAML.ORG ontologies (cf. the second last row of Table 8.3 on page 202). We use half of the average, since it is closer to the median observation and the average of the DAML.ORG ontologies also contained datatype property restrictions.

[19]This is the limes for arbitrary depths of the symmetric class tree. For depth 3 we only have 1,3 object property restrictions per class.

8.5.2.1. Knowledge Bases

Figure 8.5 depicts the extended TBox structure. The first restriction introduced on named classes is a value restriction, which restricts the value of a property to the first direct subclass. The other restriction is a number restriction. We generate the two kinds of number restrictions that can be represented in $\mathcal{L}_1^R$, i.e. $\geq 0\, P$[20] and $\leq 1\, P$. $\leq 1\, P$ is a superclass of the first and third subclass of a class, while $\geq 0\, P$ is a superclass of the second subclass of a class. We voluntarily and explicitly set a focus on the $\leq 1\, P$ restriction, which appears twice as many times as all other restrictions, since it is the restriction that allows us to deduce the equality of individuals. The size of the TBox depends on the depth of the symmetric class tree $d \geq 1$. The TBox contains the following number of class descriptions:

Constructor	Number
A	$\sum_{i=0}^{d} 3^i$
$\forall P.C$	$\sum_{i=0}^{(d-1)} 3^i$
$\leq 1\, P$	$\sum_{i=1}^{d} \frac{2*3^i}{3} = 2 * \sum_{i=0}^{(d-1)} 3^i$
$\geq 0\, P$	$\sum_{i=1}^{d} \frac{3^i}{3} = \sum_{i=0}^{(d-1)} 3^i$

We considered three variants of TBox sizes (TS,TM,TL) introduced in the previous analysis. The resulting TBox size and the number of contained descriptions is summarized in Table 8.9.

	Named	Absolute Number			Relative to Named			
Depth	Classes	$\forall P.C$	$\leq 1\, P$	$\geq 0\, P$	$\forall P.C$	$\leq 1\, P$	$\geq 0\, P$	Sum
TS	40	13	13	26	0,33	0,65	0,33	92
TM	364	121	121	242	0,33	0,66	0,33	848
TL	3280	1093	1093	2186	0,33	0,67	0,33	7652

Table 8.9.: Size and Content of Equality TBox

We did not vary the strategy for ABox generation, i.e. the three variants of ABox sizes (IS,IM,IL) introduce individuals for every *named* class. In particular, we did not explicitly assert individual equality. Our approach is rather to entail the equality of individuals. This is achieved by the presence of more than one property filler on properties which are mentioned in $\leq 1\, P$ descriptions and the algebraic axioms of equality.

[20] The reader may recall that $\geq 0\, P \equiv \top$.

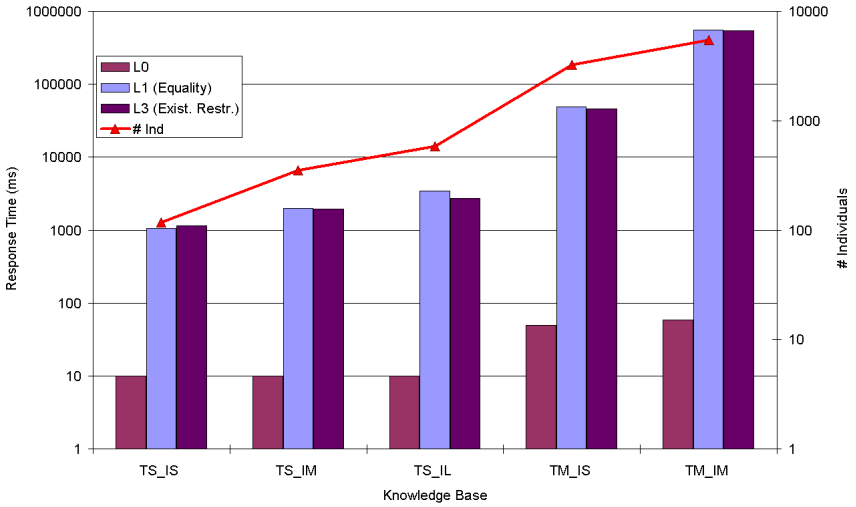


Figure 8.6.: Response Time of KAON with Equality

8.5.2.2. Results

The results of our analysis are discouraging. The introduction of equality severely deteriorates the performance of both logic databases. Figure 8.6 and Table 8.10 show that the introduction of equality slows down KAON on the order of two magnitudes on small TBoxes and three magnitudes on medium TBoxes, when compared to the performance of KAON on corresponding $\mathcal{L}_0$ knowledge bases[21]. In particular, KAON did not respond within the given 30 minute time frame on knowledge bases that were larger than the (TM-IM) variant.

KB		$\mathcal{L}_0$ (ms)	$\mathcal{L}_1$ (ms)	$\mathcal{L}_3$ (ms)	Indiv.
TS	IS	10	1051,6	1155,6	117
TS	IM	10	1974,2	1930,8	351
TS	IL	10	3442,6	2704	585
TM	IS	50	48483,8	45946	3267
TM	IM	58	554425,4	542764,4	5445

Table 8.10.: Response time of KAON with Equality

As mentioned before, XSB could not handle any dynamic programs resulting from the translation of any KB that contains equality. After switching to static programs, we could also observe the exponential blow-up of response time with XSB. However, with XSB the blow-up was even bigger than observed with KAON. Therefore,

[21] The corresponding $\mathcal{L}_0$ KB is an instantiation of the $P0$ TBox variant.

XSB could only provide timely answers for the (TS IS) and (TS IM) knowledge bases. The number of individuals severely affected the performance of XSB, as can be shown from the following table. In practise, the use of static programs instead of dynamic programs therefore only helps for the smallest knowledge base.

KB		Absolute (ms) $\mathcal{L}_1$	Absolute (ms) $\mathcal{L}_3$	Rel to KAON. $\mathcal{L}_1$	Rel to KAON. $\mathcal{L}_3$
TS	IS	1045,6	1007,4	0,99	0,87
TS	IM	203148,2	202633,2	102,9	104,95

8.5.3. Knowledge Bases with Skolem Functions

Figure 8.6 and Table 8.10 dispatch the good news that the introduction of existentials does not deteriorate performance further. We actually observe a slight improvement of performance, which can be attributed to the smaller number of $\leq 1\, P$ descriptions. For completeness, we will describe the structure of the synthetic acyclic TBoxes used in the analysis.

The synthetic TBox for analyzing the performance of $\mathcal{L}_3$ KBs varied and extended the TBox for testing equality. We now make use of three forms of number restrictions, namely $\geq 0\, P$, $\geq 1\, P$[22] and $\leq 1\, P$. We could not observe any usage of $\leq 0\, P$ in the DAML.ORG ontology library and therefore omitted this class constructor, which could nevertheless be expressed in $\mathcal{L}_3$. The analysis of the DAML.ORG ontology additionally suggests that $\exists P.C$ constitutes only 2 % of the class descriptions. We therefore instantiate the constructor on every 40th class[23]. The resulting TBox size and the number of contained descriptions is summarized in Table 8.11. The generated TBox must be acyclic such that the analysis of skolemized logic

Depth	Named Classes	Absolute Number $\forall P.C$	$\exists P.C$	$\geq 0\, P$	$\geq 1\, P$	$\leq 1\, P$	Sum
TS	40	13	1	13	13	13	93
TM	364	121	9	121	121	121	857
TL	3280	1093	82	1093	1093	1093	7734

Table 8.11.: Size and Content of TBox with Existential Restrictions

program terminates. As a result of this restriction, both constructors introducing skolem functions are instantiated on properties and classes that are not used in any other class descriptions.

[22] The reader may recall that $\geq 1\, P \equiv \exists P.\top$.

[23] This of course leads to an average of 2,5 %, but guarantees the presence of one existential restriction in the TS knowledge base variant.

8.5.4. Discussion

The presence of equality as found in Web ontologies introduces a new research problem for logic databases, which traditionally deal with equality by providing custom-tailored machinery outside of the logic program. In this "traditional" situation, which is also applied in automated theorem provers, the axiomatic system of equality is fixed and typically achieved by a dedicated, extra-logical and procedural implementation.

With Web ontologies, the axiomatic system is not fixed since new rules for deducing equality are defined by each inclusion axiom that makes use of $\leq 1\, P$ on the right-hand side of the axiom. A sound and complete optimization is yet to be found and is outside of the scope of this thesis. Our intuition is, that an optimization will be able to exploit the fact that the new equality axioms follow the strict pattern of the $\phi_{\mathcal{LP}}$ translation of $\leq 1\, P$.

An alternative to regaining acceptable performance on knowledge bases containing equality is to depart from the official semantics of number restrictions and to understand number restrictions as integrity constraints such as done in (Motik et al., 2002b). The translation of $\leq 1\, P$ would then yield the rule:

$$\texttt{:-}\, R(X,Y), R(X,Z), (Y \neq Z).$$

The rule prevents that multiple property fillers can be assigned for R. It is important to note that this translation is sound but incomplete, i.e. we do not deduce wrong information but cannot deduce all information. In consequence, any knowledge base handled by the logic database can be understood in exactly the same way by other systems that implement the official OWL semantics, but not the other way round.

A further alternative solution is presented through materialization, which allows to precompute expensive inferences such as introduced by equality. The costs of maintaining such a materialization will be analyzed in the next section.

8.6. Incremental Maintenance Analysis

This section presents our analysis of the effect of materializing knowledge bases on the performance of solving instance retrieval problems. To this extent, we measure the time needed to setup a materialization and the time needed to carry out incremental maintenance in case of changes to facts.

8.6.1. Goals and Assumptions

Our analysis particularly addresses the issue that the maintenance program contains n times[24] as many rules as the original program. Since a prototypical implementation of maintenance is only provided within KAON, we cannot evaluate XSB and Racer.

Assumptions The reader may note that the measured time is the time needed to complete each maintenance step. The maintenance typically involves two $\mathcal{QLP}$ queries for the extension of P^+ and P^- for every materialized predicate P. We assume that all predicates are materialized.

We obtain seven measures: *(a)* the time of query processing without materialization, *(b)* the time required to set up the materialization and the maintenance program, *(c)* the time required to perform maintenance when rules are added, *(d)* rules are removed, *(e)* facts are added, and *(f)* facts are removed. Finally, *(g)* assesses the time of accessing the materialized predicate.

We carried out each test using varying *Change* ratios of 10% and 15% of the facts. Facts to be removed are randomly chosen. We could only change single rules, due to limitations of the KAON Datalog engine. We give up our focus to build a knowledge base that is representative, since we have new observation targets, i.e. the above mentioned costs, for this analysis. The actual analysis was carried out in the context of (Volz et al., 2003g), which presents evaluation results on several knowledge bases.

8.6.2. Knowledge Bases

For the purpose of this chapter, we present one of the knowledge bases studied in (Volz et al., 2003g), which is very close to the P0 variant of the knowledge bases used before. However, we used 5 subclasses per class instead of 3, such that a larger number of rules was generated. Hence, every non-leaf class now has 5 subclasses. Due to the structure of the maintenance rewriting (cf. Chapter 6 on page 145), the maintenance program contains the following number of rules:

$$3 * I + 2 * E + 3 * A + 3 * L = 11 * C - 3 = 11 * \sum_{i=0}^{d} 5^i - 3$$

where C, E, I, L and A are defined as in Table 8.12.

[24] n depends on the actual structure of the rules, but is a linear factor.

C A	No. of classes No. of $C \sqsubseteq D$ axioms	$\sum_{i=0}^{d} 5^i$ C - 1
E	No. of edb predicates	C
I	No. of idb predicates	C
L	Links between P_{idb} and P_{edb}	C

Table 8.12.: Size of Maintenance

Since we were primarily interested in the observation of the costs associated with each maintenance step, we only varied the depth d of the taxonomy between the values 3,4 and 5. We did not alter the number of individuals, which was fixed to 5 as well, hence we have $5 * \sum_{i=0}^{d} 5^i$ individuals in the knowledge base.

Test		Orig.		Rule		Facts		Query
Depth	Change	Query	Setup	Remove	Add	Remove	Add	on Mat.
3	10	197	305	1386	1317	1302	1284	0
4	10	373	1347	7581	7265	7328	7310	0
5	10	1767	18391	494726	559407	631956	649123	1331
3	15	147	245	1452	1286	1301	1367	0
4	15	378	1382	7615	7308	7350	7310	0
5	15	1765	18293	273874	464588	542294	648495	1252

Table 8.13.: Costs of Materialization Procedures (in ms)

8.6.3. Results

Table 8.13 and Figure 8.7 present the average time[25] needed to solve the instance retrieval problem with and without using materialization. Additionally it shows the costs of maintenance for different types of changes (adding and removing rules and facts) and the associated cost of setting up the materialization. As we can see from the table and the figure, maintenance costs do not vary significantly with the quantity of updates, but are directly related to the size of the knowledge base.

8.6.4. Discussion

The different costs of each step in the maintenance procedure are all higher than the costs of evaluating a single query. The question whether or not to materialize is therefore determined by the application and the issue whether the system can

[25] In milliseconds on a logarithmic scale.

handle its typical workload, e.g. can it handle the intended number of users if answering a single query takes several minutes (such as in the knowledge bases which make use of equality)?

With materialization, the cost of accessing the materialized predicates can be neglected. However, the time for the analysis of the maintenance rules can be a significant bottleneck for a system, especially for large knowledge bases. For example, in one of our test runs it took almost 11 minutes to recompute the materialization after fact changes for the test with taxonomic depth 5 and 15% change of facts. Fortunately, materialization can be carried out in parallel to answering queries on top of the existing materialization. In consequence, users will have to operate on stale copies of data. Staleness of data cannot be avoided in distributed scenarios like the Web in the first place. Existing experiences, e.g. with outdated page ranks of web pages in Google, show that the quality of query answering is still good enough, if data is updated occasionally.

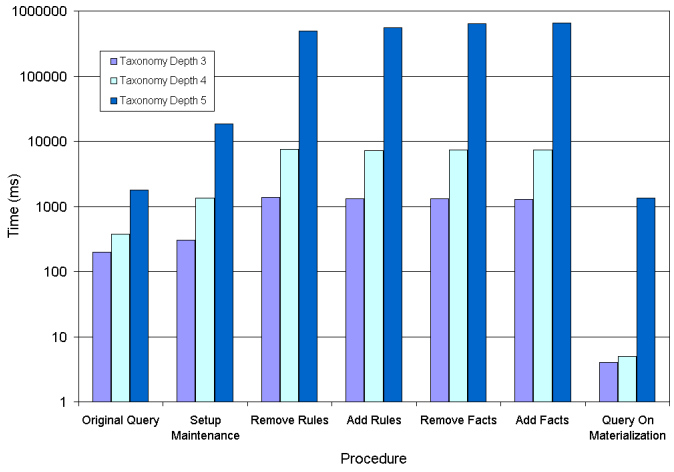


Figure 8.7.: Cost of Maintenance Procedures for 10% Change in Facts

8.7. Conclusion

Since there have been very few previous analyses of reasoning performance in total, our analysis methodology and the results of the analyses itself can be understood as a contribution. The statistical analysis of the DAML.ORG ontology library can be understood as a historic documentation of the present stage of Web ontologies.

Future analyses of Web ontology collections will almost definitely produce different average characterizations, since individual Web ontologies vary tremendously

both in size and usage of constructors. The novel insight produced by our analysis is that both the distributions of super-classes and restrictions per class follow Power Law distributions and therefore follow the Pareto law that can be observed in many other human-influenced settings as well.

While the averages calculated here might present temporary results, we nevertheless believe that our approach to derive the synthesis from average numbers is correct. For example, one can instantiate our methodology to substantiate buying decisions in an enterprise, i.e. derive averages from the collection of ontologies used in an enterprise and then follow our methodology to evaluate a particular reasoner.

The comparative performance analysis shows how drastic the improvement on ABox reasoning performance using logic databases is. We can safely say that logic databases are the only practical and scalable solution for ABox reasoning available today. The usability of such logic databases for producing correct reasoning results for the new standard Web ontology language (OWL) is the direct result of our work, which provides the necessary translations and transformations.

We can also see from the results that the current strategy of ABox reasoning applied in DL reasoners, i.e. to simulate retrieval problems by translation into satisfiability problems, does not work in practise. We conjecture that a practical solution is only possible if individuals are processed in sets, such as done in logic databases. Furthermore, first experiences with an extension (Motik et al., 2003) of our work to provide complete support for all OWL constructors appears to show that it is feasible to handle all OWL constructors using disjunctive deductive databases such as DLV (Leone et al., 2002).

The results of the expressivity analysis, however, shows that future work is necessary to provide scalable support for the more expressive variants of DLP. These optimizations will have to provide a different treatment of equality, which most likely must be handled outside of the logic program itself.

An alternative to regain scalability of query answering has been presented in our work through materialization. Finally, we were able to show through our last analysis, that it is feasible to incrementally maintaining such materializations, since the associated costs in computing time are tolerable.

Our results do not say that DL reasoners are not useful. Many applications, for example large-scale ontology engineering (Rector et al., 1999), indeed require efficient support for taxonomic reasoning. Therefore, we conjecture that a dual reasoning strategy employing logic databases and DL classifiers in parallel is needed to fully cover all aspects of Web ontology reasoning.

9. Conclusion

Treues Echo dieser Orten,
Sollt ich bei den Schmeichelworten
Süsser Leitung irrig sein?
Gib mir deine Antwort: Nein!
(Echo) *Nein!*
Oder sollte das Ermahnen,
Das so mancher Arbeit nah,
Mir die Wege besser bahnen?
Ach! so sage lieber: Ja!
(Echo) *Ja!*
5th Aria (Alto) of Cantata
"Lasst uns sorgen, lasst uns wachen"
Johann-Sebastian Bach (BWV 213)
`http://www.cs.ualberta.ca/~wfb/cantatas/213.html`

In this thesis we gave convincing arguments that the usage of Web ontology reasoning with logic databases is both useful and feasible. After a summary of the main contributions of this work, we suggest further applications for logic databases in Semantic Web applications, and conclude with an appeal for the pervasion of logic databases in modern Semantic Web applications.

9.1. Summary

The main contribution of this thesis is the enabling of Web ontology reasoning with logic databases for the new standard Web ontology language (OWL). On the theoretical side, we have shown which subset of the new standard Web ontology language can be completely supported in a correct manner. On the practical side, we have shown that logic databases allow to overcome the predominant limitation of current approaches to ABox reasoning with Description Logics such as OWL.

We take up on the subproblems formulated in the introduction and review their solutions as they were presented in this work:

- *Which fragment of OWL can be implemented in logic databases ?* We outlined a strategy to identify the fragment in Chapter 4. The strategy is based on the translation of DL axioms into first-order logic, followed by a transformation into a conjunction of Horn formulae. These formulae can be translated into the different Logic Programming languages which are supported by logic databases. While our strategy defines a general pattern that can be instantiated on arbitrary Description Logics, we instantiated the strategy on OWL to identify the fragment of OWL that can be supported. We discovered that logic databases can express almost all OWL constructors for partial class definitions. However, the constructors available for building the left-hand side of inclusions, which expresses sufficient conditions for class membership, is more restricted. Therefore the core language which can be used in class equivalence axioms is much less expressive than OWL (but more expressive than RDFS). Furthermore, we can support all OWL constructors for the definition of property characteristics.

- *What are the theoretical properties of the fragment ?* We defined Description Logic Programs (DLP) in Chapter 5 as a family of new ontology languages, which precisely contain the fragment, viz. those Description Logic constructors that can be supported by logic databases. We utilized these languages to study the theoretical properties of the fragment. In particular, we were able to show that we can solve ABox related reasoning problems, independent of the TBox, with a low polynomial complexity $O(|\mathcal{A}^{DLP}_{i\leq 2}|)^4$ in the size of the DLP ABox $\mathcal{A}^{DLP}$.

- *How can DL reasoning be reduced to queries on logic databases ?* We showed in Section 5.4 which DL reasoning problems can be reduced to queries on logic databases. Unfortunately, certain limitations have arisen from the fact that we can only retrieve individuals from $\mathcal{L}^L_i$ class descriptions. Similarly, we can only check the subsumption between $\mathcal{L}^L_i$ and $\mathcal{L}^R_i$ class descriptions. Nevertheless, we could capture an important fragment of DL queries and showed the procedure for translating DL reasoning problems into queries on logic databases. For this purpose we were able to utilize the $\phi_{\mathcal{LP}}$ function, which is used to translate DL knowledge bases into logic programs, which can be executed by logic databases.

- *How useful is the related fragment as a practical Web ontology language ?* We answered this question by showing two separate hypotheses.

  Hypothesis 1: The fragment is sufficient to express most available Web ontologies. Section 5.3 analyzed the largest currently available collection of Web ontologies and checks which fragment of those ontologies can be expressed in the

DLP languages. We were able to quantify *most* in several respects. Firstly, we showed that even the least-expressive DLP language $\mathcal{L}_0$ completely suffices to express 77% of all Web ontologies in the library. The most-expressive DLP language $\mathcal{L}_3$ could completely express 87% of all Web ontologies in the library. Secondly, we could express an average of 93% of all axioms in those ontologies in $\mathcal{L}_0$. Similarly, 99% of all axioms in the ontologies could be expressed in $\mathcal{L}_3$.

Hypothesis 2: We can efficiently and scalably reason with DLP. Our analyses in Chapter 8 showed that logic databases exhibit satisfactory performance for ABox-related reasoning problems in DLP knowledge bases that do not make use of equality reasoning. Our comparative analysis additionally showed that logic databases compare favorably with the only Description Logic reasoner, which supports ABox-reasoning today. We empirically showed, that the current approach of solving any ABox-related reasoning problem in DL reasoners, which is essentially based on simulating the retrieval problem through satisfiability checking, is impractical. DL reasoners, however, outperform logic databases in with respect to class subsumption problems. As a side-product of our performance analysis, we introduced a sophisticated methodology to obtain representative performance measures from arbitrary reasoners.

- *How can we improve the performance of ABox-related reasoning problems ?* As we could see in the expressivity performance analysis presented in Section 8.5, the introduction of equality deteriorates the performance of logic databases severely. We therefore introduced materialization as a technique to store computed entailments, such as derived through an equality theory. We showed in Chapter 6 how we can maintain such a materialization incrementally in the case of changes to the TBox and ABox. The performance analysis of our technique provided in Section 8.6 illustrated that the associated costs of incremental maintenance are tolerable, viz. that our solution is practically feasible. As a side effect, our maintenance technique, which extends previous work in the logic database context, showed how synergies between the two fields of knowledge representation - Logic Programming and Description Logics - can be established through applying techniques developed in either field for problem settings of the other.

Put together, the solutions to these problems allow scalable and practical solutions to ABox-related reasoning problems for Semantic Web applications which are based on OWL. These solutions are based on delegating those reasoning problems to logic databases. The thesis covers all necessary stages to obtain this delegation, starting from the identification of the (sub)languages for which this delegation can

be achieved, via the theoretical analysis of the properties of these languages, to the assessment of the performance obtained by the delegation. Even though logic databases have been used for Semantic Web related applications from the early phases of the Semantic Web on, no comprehensive solution, which guarantees the correctness of the obtained answers, has been presented before for OWL or its predecessors.

9.2. Further Applications

Apart from the ABox-reasoning scenario, which was the basis of this thesis, Web ontology reasoning in logic databases is helpful for further Semantic Web purposes that were identified in the introduction:

Rule Language The $\phi_{\mathcal{LP}}$ translation of DLP ontologies immediately allows users to use the target rule language of the logic database to extend the ontology with further rules. A mapping from a rule language for the Semantic Web is then as simple as a syntactic transformation into the language of the host. DLP reduces the expressiveness of the ontology language. Hence, such a rule language operating on the DLP ontology is still efficient in practise, if the rule language itself does not contain intractable language features. The reader may note that any direct and naïve rule language extension of OWL would immediately yield an undecidable language (Levy & Rousset, 1996; Schmidt-Schauß, 1989).

Query Language Similarly, logic databases typically offer more rich (but often proprietary) querying facilities and will be an interesting implementation environment for a standardized Semantic Web query language. Available research results of deductive databases present a rich source of readily available research results and can help to avoid making wrong decisions for a standard query language, such as the DL research contributions were helpful for the selection of OWL language features and could be utilized to proof the decidability of OWL DL.

Bootstrapping the Semantic Web As Section 5.3.1 shows, the convenience primitives defined for $\mathcal{L}_0$ map nicely to the core language aspects provided by object-oriented models (Cattell et al., 2000; Object Management Group, 2003), this relation can be used to bootstrap (Stojanovic et al., 2002) the Semantic Web by reusing available information models.

Legacy data integration Today, the majority of data on the Web is no longer static but resides in databases, e.g. to dynamically generate web pages. Exposing this data to the Semantic Web will require the access to the content of underlying databases (Volz et al., 2004). Description Logic reasoners such as FaCT (Horrocks et al., 1999) or Racer (Haarslev & Moller, 2001) currently require the replication of that data to enable ontology-based processing. Logic databases such as XSB (Sagonas et al., 1994), Ontobroker (Decker et al., 1999) or CORAL (Ramakrishnan et al., 1994), however, can access relational databases directly through built-in predicates. Hence, Logic Programming allows closer interaction with live data and data can remain where it can be handled most efficiently - in relational databases. Furthermore, more restricted variants of logic database languages can be directly implemented on top of SQL99-compliant relational databases (Ullman, 1988), which would increase the possible system basis further.

9.3. An Appeal

The main focus of many Semantic Web applications such as TAP[1] and AKTive Space[2] lies in ABox-related problems. Our analysis has made it clear that poor performance for ABox-related tasks in native Description Logic reasoners is a pressing problem on all levels of these applications. If the Semantic Web increases in size and more and more publicly available data will become available from more and more autonomous sources, the problem will increase in the future. To make full use of the opportunity to reason with those large amounts of data from various sources, logic databases must be applied.

Our results suggest that logic databases are currently the best solution for ABox-related OWL reasoning problems. While logic databases can only support a restricted fragment of the OWL language, we have seen that the large majority of currently available ontologies can be expressed in this fragment. These results can be understood as a critical assessment how a practical sub-language of OWL such as OWL Lite should practically look like.

For those ontologies that are not completely expressible, we only lose a small percentage of the encoded information. We can conjecture that this incompleteness is tolerable in applications, since no practical and scalable alternative exists today.

Since we were able to characterize the incompleteness of logic databases precisely, first extensions of our work (Motik et al., 2003) could directly tackle the characterized limitations and already show that we can overcome the incompleteness by

[1] `http://tap.semanticweb.org/`.
[2] `http://www.aktors.org/`.

usage of *disjunctive* database techniques.

Our conjecture for the future of Web ontology reasoning is therefore that dual reasoning strategies will evolve, which employ logic database / resolution-based techniques for ABox and DL reasoner / tableau-based techniques for TBox reasoning tasks. This dual reasoning strategy will eventually allow us to practically and scalably cover all aspects of Web ontology reasoning.

Part IV.

Appendix

A. BNF Grammar for DLP

The following grammar allows a textual specification of DLP $\mathcal{L}_3$ ontologies and has been derived from the concrete abstract syntax grammar of OWL (Bechhofer et al., 2003a). It is compatible with the (Bechhofer et al., 2003a) grammar except for the convenience primitives introduced for specifying local domains and ranges of object properties.

```
ontology ::= namespace* 'Ontology(' ontologyID? directive* ')'

directive ::= 'Annotation('
    ( ontologyPropertyID ontologyID
      | annotationPropertyID URIreference
      | annotationPropertyID dataLiteral
      | annotationPropertyID individual )
    ')'
  | fact
  | axiom

fact ::= individual
  | 'SameIndividual(' individualID individualID+ ')'
  | 'DifferentIndividuals(' individualID individualID+ ')'

individual ::= 'Individual(' individualID? annotation*
    ('type(' descriptionRight ')')* value* ')'

value ::= 'value('
    ( individualvaluedPropertyID individualID
      | individualvaluedPropertyID  individual
      | datavaluedPropertyID  dataLiteral )
  ')'

type ::= description

axiom ::= 'Class(' classID  'Deprecated'?
                'partial' annotation* descriptionRight*
                ('localdomain(' individualvaluedPropertyID descriptionRight ')')*
                ('localrange(' individualValuedPropertyID descriptionRight ')' )* )'
  | 'Class(' classID  'Deprecated'? 'complete' annotation* description* ')'
  | 'EnumeratedClass(' classID 'Deprecated'? annotation* individualID ')'
  | 'DisjointClasses(' descriptionLeft descriptionRight+ ')'
  | 'EquivalentClasses(' description description* ')'
```

```
  | 'SubClassOf(' descriptionLeft descriptionRight ')'
  | 'Datatype(' datatypeID 'Deprecated'?  annotation* )'
  | 'DatatypeProperty(' datavaluedPropertyID 'Deprecated'? annotation*
      ('super(' datavaluedPropertyID ')' )* 'Functional'?
      ('domain(' descriptionRight ')')* ('range(' dataRange ')')* ')'
  | 'ObjectProperty(' individualvaluedPropertyID 'Deprecated'? annotation*
      ( 'super(' individualvaluedPropertyID ')')*
      ( 'inverseOf(' individualvaluedPropertyID ')' )? 'Symmetric'?
      ( 'Functional' | 'InverseFunctional' | 'Transitive' )?
      ( 'domain(' descriptionRight ')' )*
      ( 'range(' descriptionRight ')' )*
     ')'
  | 'AnnotationProperty(' annotationPropertyID annotation* ')'
  | 'OntologyProperty(' ontologyPropertyID annotation* ')'
  | 'EquivalentProperties(' datavaluedPropertyID datavaluedPropertyID
      datavaluedPropertyID* ')'
  | 'EquivalentProperties(' individualvaluedPropertyID
    individualvaluedPropertyID
    individualvaluedPropertyID* ')'
  | 'SubPropertyOf(' datavaluedPropertyID  datavaluedPropertyID ')'
  | 'SubPropertyOf(' individualvaluedPropertyID
    individualvaluedPropertyID ')'


annotation ::= 'annotation('
    ( annotationPropertyID URIreference
      | annotationPropertyID dataLiteral
      | annotationPropertyID individual )
  ')'

description ::= classID
  | 'intersectionOf(' description* ')'
  | 'oneOf(' individualID ')'

descriptionLeft ::= description
  | 'intersectionOf(' descriptionLeft* ')'
  | someRestriction
  | 'unionOf(' descriptionLeft* ')'
  | 'oneOf(' individualID* ')'

descriptionRight ::= description
  | 'intersectionOf(' descriptionRight* ')'
  | 'complementOf(' descriptionRight ')'
  | allRestriction


someRestriction ::= 'restriction('
    ( datavaluedPropertyID
      dataSomeRestrictionComponent
      dataSomeRestrictionComponent*
```

```
      | individualvaluedPropertyID
      individualSomeRestrictionComponent
      individualSomeRestrictionComponent*
    )
  ')'

allRestriction ::= 'restriction('
    ( datavaluedPropertyID
      dataAllRestrictionComponent
      dataAllRestrictionComponent*
      | individualvaluedPropertyID
      individualAllRestrictionComponent
      individualAllRestrictionComponent*
    )
  ')'


dataSomeRestrictionComponent ::= 'someValuesFrom(' dataRange ')'
  | 'value(' dataLiteral ')'
  | mincardinality

individualSomeRestrictionComponent ::=  'someValuesFrom(' descriptionLeft ')'
  | 'value(' individualID ')'
  | mincardinality

dataAllRestrictionComponent ::= 'allValuesFrom(' dataRange ')'
  | 'value(' dataLiteral ')'
  | maxcardinality
  | cardinality

individualAllRestrictionComponent ::= 'allValuesFrom(' descriptionRight ')'
  | 'value(' individualID ')'
  | maxcardinality
  | cardinality

mincardinality ::= 'minCardinality(' cardNr ')'
maxcardinality ::= 'maxCardinality(' cardNr ')'
cardinality ::= 'cardinality(' cardNr ')'
cardNr ::= '0' | '1'

dataRange ::= datatypeID
  | 'rdfs:Literal'
  | 'oneOf(' dataLiteral* ')'

datatypeID ::= URIreference
classID ::= URIreference
individualID ::= URIreference
ontologyID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference
```

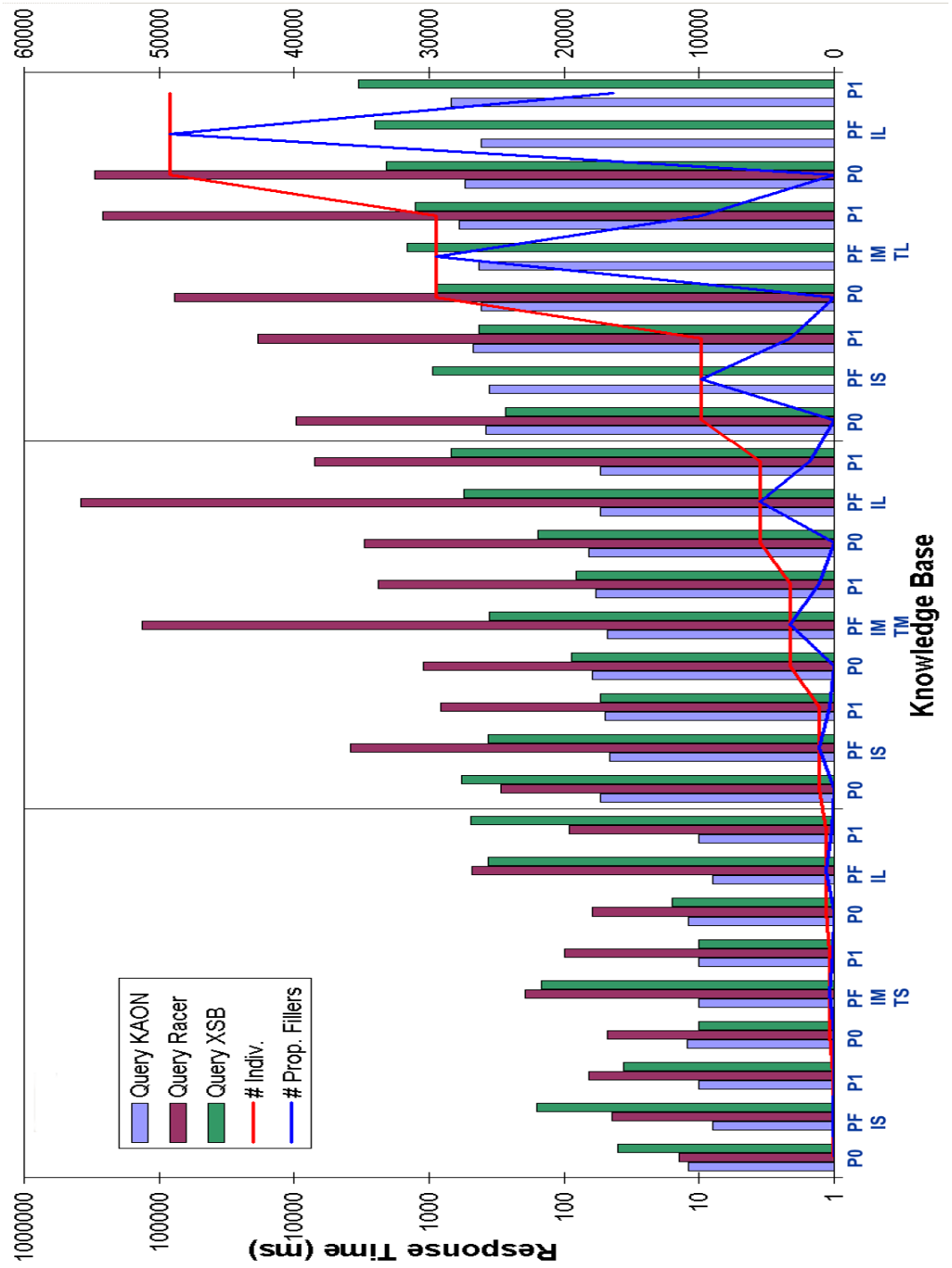
```
annotationPropertyID ::= URIreference
ontologyPropertyID ::= URIreference

/************************************************************
* The following rules are extensions to the original grammar
* provided in the
* OWL S&AS (http://www.w3.org/TR/owl-semantics/) document.
*
* They extend the grammar with a Namespace declaration and give
* a specification for URIreferences and dataLiterals.
************************************************************/

namespace ::= 'Namespace(' prefix '=' '<' absoluteURI '>' ')'

URIreference ::= '<' absoluteURI '>' | qname
qname ::= prefix ':' localname
prefix ::= letter+
localname ::= letter (letter | number | '_')*
letter ::= Any Unicode Letter
number ::= Any Unicode Number

dataLiteral ::= langString | datatypeString
langString ::= '"' string '"' ( '@' language )?
datatypeString ::= '"' string '"' '^^' URIreference
language ::= [a-z]+ ('-' [a-z0-9]+ )* encoding a language tag.

string ::= character* /* with escapes as defined below */
absoluteURI ::= character+ /* being a valid URI Reference */

/* Comments follow standard Java/C conventions, and can appear anywhere */

comment ::= '/*' character* '*/' |
            '//' (character-'\n')*

character ::= Any Unicode character

/* string escapes:
  #x0022 '"' represented as \"
  #x005C '\' represented as \\
*/
```

B. DLP Evaluation Results

B.1. Comparative Performance Results

B.1.1. Instance Retrieval

KB Variant		Test Nr.	P0 Variant KAON	P0 Variant Racer	P0 Variant XSB	PF Variant KAON	PF Variant Racer	PF Variant XSB	P1 Variant KAON	P1 Variant Racer	P1 Variant XSB
TS	IL	1	10	80	20	10	641	310	10	110	421
TS	IL	2	10	60	10	10	481	301	10	90	801
TS	IL	3	20	60	10	10	480	10	10	90	210
TS	IL	4	10	50	30	10	390	801	10	80	210
TS	IL	5	10	60	10	0	411	400	10	90	801
TS	IS	1	10	10	201	10	70	201	10	280	180
TS	IS	2	10	20	0	0	50	200	10	10	0
TS	IS	3	20	10	0	10	30	200	0	10	0
TS	IS	4	10	20	0	10	40	0	20	20	0
TS	IS	5	10	10	0	10	30	200	10	11	0
TS	IM	1	10	90	10	10	260	10	10	230	10
TS	IM	2	10	50	10	10	170	10	10	80	10
TS	IM	3	20	40	10	10	151	400	10	60	10
TS	IM	4	10	30	10	20	240	300	20	100	10
TS	IM	5	11	30	10	0	151	10	0	30	10
TM	IL	1	100	3806	241	90	1594323	230	90	8623	441
TM	IL	2	80	3155	140	50	76791	151	70	7421	1312
TM	IL	3	60	2684	140	30	70772	1102	30	7511	140
TM	IL	4	50	2994	130	70	76110	541	40	5798	1021
TM	IL	5	40	2454	130	30	74447	761	41	5678	531
TM	IS	1	101	250	30	90	6370	131	100	2323	151
TM	IS	2	60	331	1121	40	6630	30	50	431	30
TM	IS	3	40	240	711	30	1672	220	30	430	30
TM	IS	4	40	311	811	40	1422	441	40	451	30
TM	IS	5	30	331	220	30	3134	1002	30	430	30
TM	IM	1	90	1141	90	90	156876	1352	100	2453	90
TM	IM	2	60	1162	80	50	24726	90	80	2373	80
TM	IM	3	40	1052	80	30	23053	90	40	2373	80
TM	IM	4	90	1092	110	30	434484	160	40	2303	80
TM	IM	5	30	1082	81	40	23524	100	30	2424	81
TL	IL	1	691	186758	1832	601	-	1472	1062	-	2263
TL	IL	2	471	186668	2203	371	-	3014	411	-	3425
TL	IL	3	491	190464	1863	370	-	2794	721	-	3696
TL	IL	4	490	462836	2313	361	-	2593	791	-	4506
TL	IL	5	591	462395	2213	360	-	2764	420	-	2844
TL	IS	1	641	13249	270	601	-	470	631	19598	1001
TL	IS	2	330	8942	270	311	-	1162	370	18317	291
TL	IS	3	301	8753	281	291	-	1302	491	17706	290
TL	IS	4	301	9083	280	291	-	350	431	18626	291
TL	IS	5	310	8602	270	301	-	1472	461	18266	280
TL	IM	1	601	60297	861	621	-	1152	881	209512	881
TL	IM	2	360	58014	902	381	-	961	310	167621	1432
TL	IM	3	360	58074	881	381	-	1542	351	159499	2173
TL	IM	4	361	68508	841	370	-	2033	801	495272	871
TL	IM	5	360	140352	851	370	-	1562	671	266994	971


Response Time (ms)
1000000
100000
10000
1000
100
10
1
60000
50000
40000
30000
20000
10000
0
Query KAON
Query Racer
Query XSB
Indiv.
Prop. Fillers
Knowledge Base
P0 PF P1 IS
P0 PF P1 IM TS
P0 PF P1 IL
P0 PF P1 IM TM
P0 PF P1 IS
P0 PF P1 IL
P0 PF P1 IM TL
P0 PF P1 IS

B.1.2. Property Filler Retrieval

KB Variant		Test Nr.	PF Variant KAON	PF Variant Racer	PF Variant XSB	P1 Variant KAON	P1 Variant Racer	P1 Variant XSB
TS	IL	1	10	999517	0	0	53717	100
TS	IL	2	10	844825	0	10	68448	111
TS	IL	3	10	1647720	100	0	53187	0
TS	IL	4	0	1464766	0	20	46577	100
TS	IL	5	0	1468061	100	0	50072	100
TS	IS	1	10	1262	100	0	951	0
TS	IS	2	0	1862	100	10	671	110
TS	IS	3	10	1172	101	0	530	110
TS	IS	4	10	1672	0	10	521	100
TS	IS	5	10	1872	0	10	530	0
TS	IM	1	11	32196	10	0	10886	0
TS	IM	2	10	28081	0	10	10916	0
TS	IM	3	10	42511	100	0	10686	0
TS	IM	4	0	35891	100	10	10926	0
TS	IM	5	0	40257	0	10	10586	0
TM	IL	1	30	-	0	20	-	100
TM	IL	2	0	-	200	0	-	100
TM	IL	3	0	-	200	10	-	100
TM	IL	4	0	-	201	0	-	100
TM	IL	5	0	-	200	0	-	100
TM	IS	1	20	999517	0	30	405413	0
TM	IS	2	10	844825	0	0	419833	0
TM	IS	3	0	1647720	100	0	396080	0
TM	IS	4	10	1464766	100	10	386516	0
TM	IS	5	0	1468061	0	10	419694	0
TM	IM	1	60	-	0	20	-	0
TM	IM	2	10	-	0	0	-	0
TM	IM	3	0	-	0	0	-	0
TM	IM	4	10	-	0	0	-	0
TM	IM	5	0	-	0	0	-	0
TL	IL	1	20	-	1112	30	-	0
TL	IL	2	10	-	1302	20	-	100
TL	IL	3	20	-	721	20	-	100
TL	IL	4	20	-	221	30	-	100
TL	IL	5	10	-	421	20	-	100
TL	IS	1	20	-	400	30	-	0
TL	IS	2	10	-	411	30	-	0
TL	IS	3	10	-	401	30	-	0
TL	IS	4	10	-	0	30	-	0
TL	IS	5	10	-	401	30	-	0
TL	IM	1	20	-	511	30	-	100
TL	IM	2	20	-	1001	30	-	130
TL	IM	3	20	-	130	31	-	310
TL	IM	4	20	-	701	80	-	40
TL	IM	5	10	-	1002	30	-	440

B.1.3. Class Subsumption

KB Variant		Test Nr.	P0 Variant KAON	P0 Variant Racer	P0 Variant XSB	PF Variant KAON	PF Variant Racer	PF Variant XSB	P1 Variant KAON	P1 Variant Racer	P1 Variant XSB
TS	IS	1	20	0	0	30	80	100	50	0	300
TS	IS	2	20	10	0	30	0	0	50	0	301
TS	IS	3	20	0	0	31	0	0	30	0	300
TS	IS	4	30	0	0	10	0	0	41	0	0
TS	IS	5	20	0	0	10	0	210	20	0	0
TS	IM	1	10	0	0	30	0	0	90	0	0
TS	IM	2	20	0	0	30	0	0	50	0	200
TS	IM	3	10	0	0	20	0	0	50	0	300
TS	IM	4	20	10	0	10	0	0	30	0	301
TS	IM	5	10	0	100	20	0	0	40	0	301
TS	IL	1	20	0	0	30	10	300	90	0	0
TS	IL	2	20	0	0	30	0	301	50	0	100
TS	IL	3	10	0	0	20	0	0	40	0	300
TS	IL	4	20	0	0	10	0	301	30	0	300
TS	IL	5	20	0	0	20	0	301	40	0	300
TM	IS	1	150	10	300	291	0	0	201	10	100
TM	IS	2	90	0	301	110	10	0	110	0	100
TM	IS	3	60	0	300	100	10	0	90	0	301
TM	IS	4	100	10	300	120	0	0	91	0	0
TM	IS	5	90	0	300	110	10	0	100	10	100
TM	IM	1	150	10	0	210	0	0	181	0	300
TM	IM	2	121	0	0	100	0	0	100	0	0
TM	IM	3	60	0	0	100	0	0	80	0	0
TM	IM	4	60	0	0	100	0	0	80	0	0
TM	IM	5	60	0	0	100	0	0	110	0	0
TM	IL	1	150	0	0	200	10	200	170	571	0
TM	IL	2	70	0	0	160	10	0	80	0	301
TM	IL	3	50	0	0	101	0	301	70	0	0
TM	IL	4	50	10	0	101	0	301	60	0	400
TM	IL	5	50	0	0	170	0	201	60	0	301
TL	IS	1	701	0	0	2133	0	0	741	0	300
TL	IS	2	1032	0	0	1372	0	0	711	20	301
TL	IS	3	701	0	0	1252	0	0	741	0	301
TL	IS	4	701	0	0	1322	0	0	731	0	300
TL	IS	5	701	10	0	1252	0	0	721	10	201
TL	IM	1	691	10	0	1412	0	371	721	31	301
TL	IM	2	631	0	0	1563	0	421	681	0	300
TL	IM	3	631	10	0	1692	30	381	681	0	301
TL	IM	4	620	0	0	2413	371	120	701	0	301
TL	IM	5	621	10	0	1442	30	20	671	0	300
TL	IL	1	721	0	0	1592	20	301	611	0	300
TL	IL	2	691	381	0	1021	1412	0	610	10	200
TL	IL	3	711	371	120	1322	0	300	531	40	301
TL	IL	4	711	360	0	1242	30	100	551	0	300
TL	IL	5	701	370	0	1562	0	300	551	0	300

B.2. Expressivity Performance Results

Knowledge			KAON		XSB		
Knowledge		Test	Equal.	Exist.	Equal.	Exist.	Indiv.
TS	IS	1	1362	1772	1031	961	117
TS	IS	2	1062	1082	1122	971	117
TS	IS	3	1041	962	1032	1001	117
TS	IS	4	762	1011	1022	1052	117
TS	IS	5	1031	951	1021	1052	117
TS	IM	1	2243	2603	190964	187229	351
TS	IM	2	1922	1712	181551	207428	351
TS	IM	3	1852	1503	217683	219806	351
TS	IM	4	1972	1893	211755	186168	351
TS	IM	5	1882	1943	213788	212535	351
TS	IL	1	4095	2874	-	-	585
TS	IL	2	3134	3014	-	-	585
TS	IL	3	2994	2664	-	-	585
TS	IL	4	3285	2073	-	-	585
TS	IL	5	3705	2895	-	-	585
TM	IS	1	47368	43122	-	-	3267
TM	IS	2	51815	40748	-	-	3267
TM	IS	3	47578	49602	-	-	3267
TM	IS	4	41169	47408	-	-	3267
TM	IS	5	54489	48850	-	-	3267
TM	IM	1	397732	333359	-	-	5445
TM	IM	2	618690	570220	-	-	5445
TM	IM	3	550261	593413	-	-	5445
TM	IM	4	792500	601265	-	-	5445
TM	IM	5	412944	615565	-	-	5445

B.3. Materialization

Test	D	NS	NI	P	Change	Orig Query			Query
						Orig Average	Minimum	Maximum	Materializatio
Taxonomy	3	5	5	0	10	197	80	491	0
Taxonomy	4	5	5	0	10	373	290	571	0
Taxonomy	5	5	5	0	10	1767	1482	2463	1331
Taxonomy	3	5	5	0	15	147	60	311	0
Taxonomy	4	5	5	0	15	378	280	581	0
Taxonomy	**5**	**5**	**5**	**0**	**15**	**1765**	**1373**	**2464**	**1252**

Test	D	NS	NI	P	Change	Setup Maintenance		
						Average	Minimum	Maximum
Taxonomy	3	5	5	0	10	305	200	441
Taxonomy	4	5	5	0	10	1347	1212	1622
Taxonomy	5	5	5	0	10	18391	16694	19318
Taxonomy	3	5	5	0	15	245	141	251
Taxonomy	4	5	5	0	15	1382	1232	1683
Taxonomy	**5**	**5**	**5**	**0**	**15**	**18293**	**16714**	**19017**

Test	D	NS	NI	P	Change	Removing Rules		
						Average	Minimum	Maximum
Taxonomy	3	5	5	0	10	1386	1292	1592
Taxonomy	4	5	5	0	10	7581	7291	8352
Taxonomy	5	5	5	0	10	494726	227747	717452
Taxonomy	3	5	5	0	15	1452	1292	1772
Taxonomy	4	5	5	0	15	7615	7330	8372
Taxonomy	**5**	**5**	**5**	**0**	**15**	**273874**	**189933**	**386005**

Test	D	NS	NI	P	Change	Adding Rules		
						Average	Minimum	Maximum
Taxonomy	3	5	5	0	10	1317	1252	1463
Taxonomy	4	5	5	0	10	7265	7240	7290
Taxonomy	5	5	5	0	10	559407	393666	706696
Taxonomy	3	5	5	0	15	1286	1251	1332
Taxonomy	4	5	5	0	15	7308	7291	7331
Taxonomy	**5**	**5**	**5**	**0**	**15**	**464588**	**247826**	**611980**

Test	D	NS	NI	P	Change	Removing Facts		
						Average	Minimum	Maximum
Taxonomy	3	5	5	0	10	1302	1282	1332
Taxonomy	4	5	5	0	10	7328	7281	7361
Taxonomy	5	5	5	0	10	631956	487551	759261
Taxonomy	3	5	5	0	15	1301	1291	1312
Taxonomy	4	5	5	0	15	7350	7340	7371
Taxonomy	**5**	**5**	**5**	**0**	**15**	**542294**	**381628**	**650265**

Test	D	NS	NI	P	Change	Adding Facts		
						Average	Minimum	Maximum
Taxonomy	3	5	5	0	10	1284	1262	1312
Taxonomy	4	5	5	0	10	7310	7270	7380
Taxonomy	5	5	5	0	10	649123	522761	781173
Taxonomy	3	5	5	0	15	1367	1282	1612
Taxonomy	4	5	5	0	15	7310	7271	7350
Taxonomy	**5**	**5**	**5**	**0**	**15**	**648495**	**576319**	**756978**

C. DAML Ontology Library in DLP

C.1. Individual Evaluation Results

URI	# Axioms (Absolute)					# Axioms (Relative[1])			
	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	OWL	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$
http://daml.umbc.edu/ontologies/cobra/0.3/agent	19	0	0	0	0	100%	0%	0%	0%
http://daml.umbc.edu/ontologies/ittalks/event	1	0	0	0	0	100%	0%	0%	0%
http://daml.umbc.edu/ontologies/ittalks/person	1	0	0	0	0	100%	0%	0%	0%
http://daml.umbc.edu/ontologies/ittalks/talk	2	0	0	0	0	100%	0%	0%	0%
http://daml.umbc.edu/ontologies/profile-ont	7	0	0	0	0	100%	0%	0%	0%
http://daml.umbc.edu/ontologies/talk-ont	4	0	0	0	0	100%	0%	0%	0%
http://derpi.tuwien.ac.at/ andrei/cerif-rdf-dc-mn.daml	230	0	10	121	1	63,54%	0%	2,76%	33,43%
http://edge.mcs.drexel.edu/MUG/2001/05/16/sbf.daml	266	0	0	0	0	100%	0%	0%	0%
http://jupiter.tezu.ernet.in/ontology/tumca.daml	15	0	1	0	1	88,24%	0%	5,88%	0%
http://ksl.stanford.edu/projects/DAML/chimaera-jtp-cardinality-test1.daml	13	1	0	0	1	86,67%	6,67%	0%	0%
http://loki.cae.drexel.edu/ how/HydrologicUnits/2003/09/hu#	4	0	0	0	0	100%	0%	0%	0%
http://mr.teknowledge.com/DAML/Imaging.daml	2	2	0	0	0	50%	50%	0%	0%
http://onto.cs.yale.edu:8080/ontologies/wsdl-ont.daml	2	0	0	2	0	50%	0%	0%	50%
http://onto.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRSTR.daml	1450	0	0	0	0	100%	0%	0%	0%
http://ontolingua.stanford.edu/doc/chimaera/ontologies/wines.daml	873	36	39	41	49	84,1%	3,47%	3,76%	3,95%
http://opencyc.sourceforge.net/daml/cyc-transportation.daml	526	1	26	0	0	95,12%	0,18%	4,7%	0%
http://orlando.drc.com/daml/ontology/									
.. Bibliographic/current/	12	0	0	0	0	100%	0%	0%	0%
.. Condition/UJTL/v4.0/current/	3	0	0	0	0	100%	0%	0%	0%
.. Locator/current/	2	0	0	0	0	100%	0%	0%	0%
.. Organization/current/	3	0	0	0	0	100%	0%	0%	0%
.. TaskList/current/	3	0	0	0	0	100%	0%	0%	0%
.. TaskListUJTLScenario/current/	4	0	12	0	0	25%	0%	75%	0%
.. Thesaurus/CALL/current/	11	0	0	0	0	100%	0%	0%	0%
http://orlando.drc.com/SemanticWeb/DAML/Ontology/									
.. Goal-Objective	2	0	0	0	0	100%	0%	0%	0%
.. NationalSecurity	5	0	0	0	0	100%	0%	0%	0%
http://orlando.drc.com/SemanticWeb/OWL/Ontology/									
.. Contact/ver/1.0.0/Contact-ont.owl	29	12	0	0	0	70,73%	29,27%	0%	0%
.. Holiday/ver/1.0.0/Holiday-ont.owl	6	2	0	2	0	60%	20%	0%	20%
.. TimeZone/ver/1.0.0/TimeZone-ont.owl	5	5	0	0	0	50%	50%	0%	0%
http://purl.org/rss/1.0/	4	0	0	0	0	100%	0%	0%	0%
http://reliant.teknowledge.com/DAML/									
.. ATO98MessageSet_Ontology.owl	50	0	0	0	0	100%	0%	0%	0%
.. ATO_Mission_Models.owl	199	0	0	0	0	100%	0%	0%	0%
.. ATO_Ontology.owl	513	0	0	0	0	100%	0%	0%	0%
.. Communications.owl	193	0	0	0	0	100%	0%	0%	0%
.. Economy.owl	1050	0	0	0	0	100%	0%	0%	0%
.. Elements.owl	118	0	0	0	0	100%	0%	0%	0%
.. Government.owl	1574	0	0	0	0	100%	0%	0%	0%
.. Mid-level-ontology.owl	2004	0	0	0	0	100%	0%	0%	0%
.. Military.owl	62	0	0	0	0	100%	0%	0%	0%
.. SUMO.owl	1831	0	0	0	0	100%	0%	0%	0%
.. Terrorists.owl	177	0	0	0	0	100%	0%	0%	0%
.. WMD.owl	464	0	0	0	0	100%	0%	0%	0%
http://siul02.si.ehu.es/Aingeru/OperOnt.owl	165	6	24	24	2	74,66%	2,71%	10,86%	10,86%
http://taga.umbc.edu/ontologies/fipaowl	10	0	0	0	0	100%	0%	0%	0%
http://taga.umbc.edu/ontologies/fipaowl.owl	156	0	0	0	0	100%	0%	0%	0%
http://ubot.lockheedmartin.com/ubot/2001/08/									
.. baby-shoe/shoeproj-ont.daml	4	0	0	0	0	100%	0%	0%	0%
.. extraction-ont.daml	6	0	0	0	0	100%	0%	0%	0%
.. ubot-ont.daml	66	0	0	0	0	100%	0%	0%	0%
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Time.daml	10	0	0	0	0	100%	0%	0%	0%
http://www.aktors.org/ontology/portal	166	0	0	0	0	100%	0%	0%	0%

URI	# Axioms (Absolute)					# Axioms (Relative[2])			
	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{SHINO}$	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$
http://www.cs.man.ac.uk/ horrocks/Ontologies/									
..tambis.daml	370	0	0	0	0	100%	0%	0%	0%
http://www.cs.man.ac.uk/ horrocks/OWL/Ontologies/									
.. galen.owl	3197	150	0	1895	0	60,99%	2,86%	0%	36,15%
.. ka.owl	216	0	0	0	0	100%	0%	0%	0%
.. mad_cows.owl	80	1	4	17	7	73,39%	0,92%	3,67%	15,6%
http://www.cs.man.ac.uk/ lopatena/cerif/cerif.daml	351	12	6	137	0	69,37%	2,37%	1,19%	27,08%
http://www.cs.umbc.edu/ yzou1/daml/acl.daml	5	1	0	0	1	71,43%	14,29%	0%	0%
http://www.cs.umbc.edu/ yzou1/daml/acldaml.daml	2	1	0	0	1	50%	25%	0%	0%
http://www.cs.umd.edu/ golbeck/daml/running.daml	1	0	0	0	0	100%	0%	0%	0%
http://www.cs.umd.edu/ golbeck/daml/vegetarian.daml	15	0	0	0	0	100%	0%	0%	0%
http://www.cs.yale.edu/ dvm/daml/drsonto.daml	60	0	0	0	1	98,36%	0%	0%	0%
http://www.cse.dmu.ac.uk/ monika/Pages/									
.. Ontologies/CinemaAndMovies.daml	26	2	0	0	2	86,67%	6,67%	0%	0%
http://www.daml.org									
../2001/01/gedcom/gedcom	6	0	0	0	0	100%	0%	0%	0%
../2001/02/geofile/geofile-ont	106	0	0	0	0	100%	0%	0%	0%
../2001/02/projectplan/projectplan	21	0	0	0	0	100%	0%	0%	0%
../2001/06/itinerary/itinerary-ont	4	0	0	0	0	100%	0%	0%	0%
../2001/06/map/map-ont	6	0	0	0	0	100%	0%	0%	0%
../2001/08/baseball/baseball-ont	100	0	0	0	0	100%	0%	0%	0%
../2001/10/agenda/agenda-ont	7	0	0	0	0	100%	0%	0%	0%
../2001/10/cvslog/cvslog-ont	2	0	0	0	0	100%	0%	0%	0%
../2001/10/office/office	4	0	0	0	0	100%	0%	0%	0%
../2002/02/chiefs/chiefs-ont	1	0	0	0	0	100%	0%	0%	0%
../2002/03/agents/agent-ont	16	0	0	0	0	100%	0%	0%	0%
../2002/03/agents/mcda	3	0	0	0	0	100%	0%	0%	0%
../2002/03/darpadir/darpadir-ont	2	0	0	0	0	100%	0%	0%	0%
../2002/04/geonames/geonames-ont	1	0	0	0	0	100%	0%	0%	0%
../2002/05/mcda/mcda-ont	8	0	0	0	0	100%	0%	0%	0%
../2002/10/units/units-ont	3	0	0	0	0	100%	0%	0%	0%
../2002/11/jbi/jbi-ont	2	0	0	0	0	100%	0%	0%	0%
../2003/01/movienight/movienight-ont	4	0	0	0	0	100%	0%	0%	0%
../2003/02/usps/usps-ont.owl	61	0	0	0	0	100%	0%	0%	0%
../2003/04/agents/enpmap	1	0	0	0	0	100%	0%	0%	0%
../2003/05/subway/subway-ont	4	0	0	0	0	100%	0%	0%	0%
../2003/09/factbook/factbook-ont	35	0	0	0	0	100%	0%	0%	0%
../projects/integration/projects-20010811	13	0	0	0	0	100%	0%	0%	0%
../tools/tools-ont	5	0	0	0	0	100%	0%	0%	0%
http://www.daml.ri.cmu.edu/ont/AirportCodes.daml	4	0	0	0	0	100%	0%	0%	0%
http://www.daml.ri.cmu.edu/ont/homework/atlas-date.daml	14	2	0	0	2	77,78%	11,11%	0%	0%
http://www.daml.ri.cmu.edu/ont/homework/									
.. cmu-ri-center-ont.daml	18	0	0	0	0	100%	0%	0%	0%
.. cmu-ri-courses-ont.daml	15	0	1	0	1	88,24%	0%	5,88%	0%
.. cmu-ri-employmenttypes-ont.daml	29	0	0	0	0	100%	0%	0%	0%
.. cmu-ri-labgroup-ont.daml	22	0	0	0	0	100%	0%	0%	0%
.. cmu-ri-people-ont.daml	21	0	0	0	0	100%	0%	0%	0%
.. cmu-ri-project-ont.daml	20	0	0	0	0	100%	0%	0%	0%
.. cmu-ri-publications-ont.daml	84	0	91	0	1	47,73%	0%	51,7%	0%
http://www.daml.ri.cmu.edu/ont/USCity.daml	4	0	0	0	0	100%	0%	0%	0%
http://www.daml.ri.cmu.edu/ont/USRegionState.daml	398	0	0	0	0	100%	0%	0%	0%
http://www.isi.edu/webscripter/communityreview/									
.. abstract-review-o	6	3	0	2	1	50%	25%	0%	16,67%
.. scientific-review-o	1	0	0	0	0	100%	0%	0%	0%
http://www.isi.edu/webscripter/todo.o.daml	29	0	0	0	0	100%	0%	0%	0%
http://www.kestrel.edu/DAML/2000/12/TIME.daml	10	0	0	0	0	100%	0%	0%	0%
http://www.ksl.stanford.edu/projects/DAML/									
..ksl-daml-desc.daml	102	0	0	0	0	100%	0%	0%	0%
..UNSPSC.daml	9795	0	0	0	0	100%	0%	0%	0%
http://www.mindswap.org/2003/CancerOntology/									
.. nciOncology.owl	32979	0	0	13961	0	70,26%	0%	0%	29,74%
http://www.site.uottawa.ca/ mkhedr/									
..Context.daml	83	0	0	0	0	100%	0%	0%	0%
..Context2.daml	83	0	0	0	0	100%	0%	0%	0%
..ContextDependancy.daml	29	0	2	0	0	93,55%	0%	6,45%	0%
..NewFuzzy.owl	66	1	9	3	6	77,65%	1,18%	10,59%	3,53%
http://www.tridedalo.com.br/2003/07/umls/	135	0	0	0	0	100%	0%	0%	0%
http://www.w3.org/2000/10/annotation-ns#	4	0	0	0	0	100%	0%	0%	0%
http://www.w3.org/2000/10/annotationType#	7	0	0	0	0	100%	0%	0%	0%
http://www.w3.org/2000/10/swap/pim/contact.rdf	23	0	0	0	0	100%	0%	0%	0%
http://www.w3.org/2000/10/swap/pim/doc.rdf	24	0	0	0	0	100%	0%	0%	0%
http://www.w3.org/2001/05/rdf-ds/datastore-schema	3	0	0	0	0	100%	0%	0%	0%

[2]Aggregated $\Sigma_{j \leq i} \mathcal{L}_j$

C.1.1. Converted DAML+OIL Ontologies

The following ontologies have been converted to OWL by means of the the OilEd tool (CVS Version of August 20, 2003). The correctness of the conversion has been assessed empirically, but is not guaranteed.

http://daml.umbc.edu/ontologies/classification.daml
http://daml.umbc.edu/ontologies/ittalks/address
http://daml.umbc.edu/ontologies/ittalks/assertions
http://daml.umbc.edu/ontologies/cobra/0.3/agent
http://daml.umbc.edu/ontologies/ittalks/event
http://daml.umbc.edu/ontologies/ittalks/person
http://daml.umbc.edu/ontologies/ittalks/talk
http://daml.umbc.edu/ontologies/ittalks/topic
http://daml.umbc.edu/ontologies/profile-ont
http://daml.umbc.edu/ontologies/talk-ont
http://derpi.tuwien.ac.at/~andrei/cerif-rdf-dc-mn.daml
http://edge.mcs.drexel.edu/MUG/2001/05/16/sbf.daml
http://jupiter.tezu.ernet.in/ontology/tumca.daml
http://ksl.stanford.edu/projects/DAML/chimaera-jtp-cardinality-test1.daml
http://loki.cae.drexel.edu/~how/HydrologicUnits/2003/09/hu
http://mr.teknowledge.com/DAML/Imaging.daml
http://mr.teknowledge.com/DAML/pptOntology.daml
http://onto.cs.yale.edu:8080/ontologies/wsdl-ont.daml
http://onto.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRDEF.daml
http://onto.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRSTR.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_I1.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_I2.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_I3.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_I4.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_II1.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_II2.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_II3.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_II4.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_III1.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_III2.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_III3.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_III4.daml
http://ontolingua.stanford.edu/doc/chimaera/ontologies/wines.daml
http://opencyc.sourceforge.net/daml/cyc-transportation.daml
http://opencyc.sourceforge.net/daml/naics
http://orlando.drc.com/daml/ontology/Bibliographic/current/
http://orlando.drc.com/daml/Ontology/Commercial/Shipping/current/
http://orlando.drc.com/daml/Ontology/Condition/UJTL/v4.0/current/
http://orlando.drc.com/daml/ontology/Fugitive/current/
http://orlando.drc.com/daml/Ontology/Genealogy/current/
http://orlando.drc.com/daml/ontology/Glossary/current/
http://orlando.drc.com/daml/Ontology/GPS/Coordinates/current/
http://orlando.drc.com/daml/Ontology/Intelligence/Report/current/
http://orlando.drc.com/daml/ontology/Locator/current/
http://orlando.drc.com/daml/ontology/Organization/current/
http://orlando.drc.com/daml/ontology/Person/current/
http://orlando.drc.com/daml/Ontology/POC/current/
http://orlando.drc.com/daml/ontology/TaskList/current/
http://orlando.drc.com/daml/Ontology/TaskListUJTLScenario/current/
http://orlando.drc.com/daml/Ontology/Thesaurus/CALL/current/
http://orlando.drc.com/daml/ontology/UniversalProperty/current/
http://orlando.drc.com/SemanticWeb/DAML/Ontology/dc
http://orlando.drc.com/SemanticWeb/DAML/Ontology/DIS/Entity/Platform/Land
http://orlando.drc.com/SemanticWeb/DAML/Ontology/Goal-Objective
http://orlando.drc.com/SemanticWeb/DAML/Ontology/NationalSecurity
http://orlando.drc.com/SemanticWeb/OWL/Ontology/spaceshuttle/crew
http://orlando.drc.com/SemanticWeb/OWL/Ontology/spaceshuttle/mission
http://orlando.drc.com/semanticweb/owl/ontology/spaceshuttle/system
http://purl.org/net/swn
http://purl.org/rss/1.0/
http://reliant.teknowledge.com/DAML/terroristAttackTypes.daml
http://taga.umbc.edu/ontologies/fipaowl
http://ubot.lockheedmartin.com/ubot/2001/08/baby-shoe/shoeproj-ont.daml
http://ubot.lockheedmartin.com/ubot/2001/08/extraction-ont.daml
http://ubot.lockheedmartin.com/ubot/2001/08/ubot-ont.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Awards.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Bio.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/ContactInfo.daml

http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Course.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Date.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Image.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Organization.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Person.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Project.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Publication.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Researcher.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Time.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Topic.daml
http://www.aktors.org/ontology/portal
http://www.civil.auc.dk/~i6ycl/itcode/test/itcode-projectteam.rdf
http://www.cs.man.ac.uk/~horrocks/Ontologies/tambis.daml
http://www.cs.man.ac.uk/~lopatena/cerif/cerif.daml
http://www.cs.umbc.edu/~yzou1/daml/acl.daml
http://www.cs.umbc.edu/~yzou1/daml/acldaml.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/beer1.0.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/cs1.0.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/docmnt1.0.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/general1.0.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/personal1.0.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/tseont.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/univ1.0.daml
http://www.cs.umd.edu/~golbeck/daml/baseball.daml
http://www.cs.umd.edu/~golbeck/daml/running.daml
http://www.cs.umd.edu/~golbeck/daml/vegetarian.daml
http://www.cs.yale.edu/~dvm/daml/agent-ont.daml
http://www.cs.yale.edu/~dvm/daml/bib-ont.daml
http://www.cs.yale.edu/~dvm/daml/drsonto.daml
http://www.cs.yale.edu/~dvm/daml/exp-ont.daml
http://www.cs.yale.edu/~dvm/daml/pddlonto.daml
http://www.cse.dmu.ac.uk/~monika/Pages/Ontologies/CinemaAndMovies.daml
http://www.daml.ecs.soton.ac.uk/ont/currency.daml
http://www.daml.org/2000/10/daml-ont
http://www.daml.org/2001/01/gedcom/gedcom
http://www.daml.org/2001/02/geofile/geofile-ont
http://www.daml.org/2001/02/projectplan/projectplan
http://www.daml.org/2001/06/expenses/amex-ont
http://www.daml.org/2001/06/expenses/check-ont
http://www.daml.org/2001/06/expenses/eecr-ont
http://www.daml.org/2001/06/expenses/trip-ont
http://www.daml.org/2001/06/itinerary/itinerary-ont
http://www.daml.org/2001/06/map/map-ont
http://www.daml.org/2001/08/baseball/baseball-ont
http://www.daml.org/2001/09/countries/fips-10-4-ont
http://www.daml.org/2001/09/countries/iso-3166-ont
http://www.daml.org/2001/10/agenda/agenda-ont
http://www.daml.org/2001/10/cvslog/cvslog-ont
http://www.daml.org/2001/10/html/airport-ont
http://www.daml.org/2001/10/html/nyse-ont
http://www.daml.org/2001/10/html/zipcode-ont
http://www.daml.org/2001/10/office/office
http://www.daml.org/2002/02/chiefs/chiefs-ont
http://www.daml.org/2002/02/telephone/1/areacodes-ont
http://www.daml.org/2002/03/agents/agent-ont
http://www.daml.org/2002/03/agents/mcda
http://www.daml.org/2002/03/darpadir/darpadir-ont
http://www.daml.org/2002/03/metrics/metrics-ont
http://www.daml.org/2002/03/ranks/rank-ont
http://www.daml.org/2002/03/usnships/ship-ont
http://www.daml.org/2002/04/classification/classification-ont
http://www.daml.org/2002/04/geonames/geonames-ont
http://www.daml.org/2002/05/mcda/mcda-ont
http://www.daml.org/2002/08/nasdaq/nasdaq-ont
http://www.daml.org/2002/09/milservices/milservices-ont
http://www.daml.org/2002/10/sndl/unit-ont
http://www.daml.org/2002/10/units/units-ont
http://www.daml.org/2002/11/jbi/jbi-ont
http://www.daml.org/2003/01/movienight/movienight-ont
http://www.daml.org/2003/04/agents/enpmap
http://www.daml.org/2003/05/subway/subway-ont
http://www.daml.org/2003/09/factbook/factbook-ont
http://www.daml.org/projects/integration/projects-20010811
http://www.daml.org/tools/tools-ont
http://www.daml.ri.cmu.edu/ont/AirportCodes.daml
http://www.daml.ri.cmu.edu/ont/homework/atlas-date.daml
http://www.daml.ri.cmu.edu/ont/homework/atlas-publications.daml
http://www.daml.ri.cmu.edu/ont/homework/cmu-ri-center-ont.daml

```
http://www.daml.ri.cmu.edu/ont/homework/cmu-ri-courses-ont.daml
http://www.daml.ri.cmu.edu/ont/homework/cmu-ri-employmenttypes-ont.daml
http://www.daml.ri.cmu.edu/ont/homework/cmu-ri-labgroup-ont.daml
http://www.daml.ri.cmu.edu/ont/homework/cmu-ri-people-ont.daml
http://www.daml.ri.cmu.edu/ont/homework/cmu-ri-project-ont.daml
http://www.daml.ri.cmu.edu/ont/homework/cmu-ri-publications-ont.daml
http://www.daml.ri.cmu.edu/ont/USCity.daml
http://www.daml.ri.cmu.edu/ont/USRegionState.daml
http://www.davincinetbook.com:8080/daml/rdf/homework3/projectGutenbergOnt.daml
http://www.isi.edu/webscripter/bibtex.o.daml
http://www.isi.edu/webscripter/communityreview/abstract-review-o
http://www.isi.edu/webscripter/communityreview/scientific-review-o
http://www.isi.edu/webscripter/division.o.daml
http://www.isi.edu/webscripter/document.o.daml
http://www.isi.edu/webscripter/event.o.daml
http://www.isi.edu/webscripter/person.o.daml
http://www.isi.edu/webscripter/project.o.daml
http://www.isi.edu/webscripter/publication.o.daml
http://www.isi.edu/webscripter/snapshot.o.daml
http://www.isi.edu/webscripter/todo.o.daml
http://www.kestrel.edu/DAML/2000/12/CAPACITY.daml
http://www.kestrel.edu/DAML/2000/12/DEMAND.daml
http://www.kestrel.edu/DAML/2000/12/instances.daml
http://www.kestrel.edu/DAML/2000/12/OPERATION.daml
http://www.kestrel.edu/DAML/2000/12/RESOURCE.daml
http://www.kestrel.edu/DAML/2000/12/TIME.daml
http://www.ksl.stanford.edu/projects/DAML/ksl-daml-desc.daml
http://www.ksl.stanford.edu/projects/DAML/UNSPSC.daml
http://www.semanticweb.org/library/wordnet/wordnet-20000620.rdfs
http://www.semanticweb.org/ontologies/swrc-onto-2000-09-10.daml
http://www.site.uottawa.ca/~mkhedr/Context.daml
http://www.site.uottawa.ca/~mkhedr/Context2.daml
http://www.site.uottawa.ca/~mkhedr/ContextDependancy.daml
http://www.site.uottawa.ca/~mkhedr/ContextFinal2.daml
http://www.tridedalo.com.br/2003/07/umls/
http://www.w3.org/2000/10/annotation-ns
http://www.w3.org/2000/10/annotationType
http://www.w3.org/2000/10/swap/pim/contact.rdf
http://www.w3.org/2001/03/thread
http://www.w3.org/2000/10/swap/pim/doc.rdf
http://www.w3.org/2001/05/rdf-ds/datastore-schema
```

C.1.2. Excluded Ontologies

The following ontologies could not be included in the analysis due to several types of errors. 32 ontologies could not be found at the specified URL, 5 ontologies could not be parsed, 23 ontologies were not OWL DL ontologies, 72 ontologies could not be converted from DAML+OIL to OWL.

Parsing Failure

```
http://orlando.drc.com/SemanticWeb/OWL/Ontology/Contact/ver/1.0.0/Contact-ont.owl
http://projects.teknowledge.com/DAML/DynamicOntology1.owl
http://projects.teknowledge.com/DAML/Ontology.owl
http://reliant.teknowledge.com/DAML/Geography.owl
http://reliant.teknowledge.com/DAML/Transportation.owl
```

HTTP 404 Error (Not Found)

```
http://cim4.ie.psu.edu:12/daml/rios/2001/05/RIOS\_Operation.daml
http://cim4.ie.psu.edu:12/daml/rios/2001/05/RIOS\_Operation_requirement.daml
http://cim4.ie.psu.edu:12/daml/rios/2001/05/RIOS\_Process.daml
http://cim4.ie.psu.edu:12/daml/rios/2001/05/RIOS\_process_requirement.daml
http://cim4.ie.psu.edu:12/daml/rios/2001/05/ServiceDescription.daml
```

http://cse.seu.edu.cn/people/ysdong/kqml.owl
http://daml.umbc.edu/ontologies/calendar-ont.daml
http://daml.umbc.edu/ontologies/cobra/0.4/action.owl
http://daml.umbc.edu/ontologies/cobra/0.4/adjustlighting.owl
http://daml.umbc.edu/ontologies/cobra/0.4/space-basic.owl
http://daml.umbc.edu/ontologies/cobra/0.4/time-basic.owl
http://daml.umbc.edu/ontologies/dreggie-ont.daml
http://daml.umbc.edu/ontologies/topic-ont.daml
http://daml.umbc.edu/ontologies/trust-ont.daml
http://grcinet.grci.com/maria/www/codipsite/Onto/DublinCore/DublinCore_V27Aug2001.daml
http://grcinet.grci.com/maria/www/codipsite/Onto/Project/ProjectOntology_V26Jul2001.daml
http://grcinet.grci.com/maria/www/codipsite/Onto/TMD/TMDOntology_V27Aug2001.daml
http://grcinet.grci.com/maria/www/codipsite/Onto/WebDirectory/WebDirectory_V27Aug2001.daml
http://grcinet.grci.com/maria/www/CodipSite/Onto/WebSite/WebSiteOntology_V27Aug2001.daml
http://hjkhasdjkasljd
http://jupiter.tezu.ernet.in/ontology/mca.daml
http://mnemosyne.umd.edu/~aelkiss/daml/serial1.2.daml
http://mnemosyne.umd.edu/~aelkiss/weather-ont.daml
http://mr.teknowledge.com/daml/Homeworks/HomeWork1/ResearchProjectOntology.daml
http://mr.teknowledge.com/daml/homeworks/HomeWork3/BriefingOntology.daml
http://mr.teknowledge.com/daml/homeworks/HomeWork3/SurveyOntology.daml
http://mr.teknowledge.com/daml/ontologies/ImageFingerprinting/2001/04/BriefingsOntology.daml
http://mr.teknowledge.com/daml/ontologies/ImageFingerprinting/2001/04/ImageFingerprintingOntology-web.daml
http://mr.teknowledge.com/daml/ontologies/ImageFingerprinting/2001/04/ImageFingerprintsOntology-briefings.daml
http://www.cyc.com/cyc-2-1/cyc-vocab.daml
http://www.engr.sc.edu/research/cit/projects/DAML.html
http://www.lgi2p.ema.fr/~ranwezs/ontologies/musicV1.0.daml
http://www.lgi2p.ema.fr/~ranwezs/ontologies/soccerV2.0.daml

HTTP Error 403 - No Access Rights

http://horus.isx.com/markup/2002/01/countries2.rdf
http://horus.isx.com/onts/2001/12/draft/horuslocusont.daml
http://isx.com/~phaglic/horus/daml/onts/englishpubont.daml
http://www.cse.sc.edu/~dukle/ontologies/football-ont.daml
http://www.daml.org/ontologies/ontologies-ont

DAML+OIL to OWL Conversion Errors

http://cicho0.tripod.com/cs_Courses_ont
http://cicho0.tripod.com/cs_LecturingStaff_ont
http://cicho0.tripod.com/cs_Staff_ont
http://cicho0.tripod.com/Dep_of_Computer_Science
http://daml.umbc.edu/ontologies/classification.daml
http://daml.umbc.edu/ontologies/ittalks/address
http://daml.umbc.edu/ontologies/ittalks/assertions
http://daml.umbc.edu/ontologies/ittalks/topic
http://mr.teknowledge.com/DAML/ArtOntology.daml
http://mr.teknowledge.com/DAML/pptOntology.daml
http://onto.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRDEF.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_I1.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_I2.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_I3.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_I4.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_II1.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_II2.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_II3.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_II4.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_III1.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_III2.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_III3.daml
http://ontobroker.semanticweb.org/ontos/compontos/tourism_III4.daml
http://ontolingua.stanford.edu/doc/chimaera/ontologies/world-fact-book.daml
http://opencyc.sourceforge.net/daml/cyc.daml
http://orlando.drc.com/daml/Ontology/Commercial/Shipping/current/
http://orlando.drc.com/daml/Ontology/DAML-extension/current/
http://orlando.drc.com/daml/ontology/DC/current/
http://orlando.drc.com/daml/ontology/Fugitive/current/
http://orlando.drc.com/daml/Ontology/GPS/Coordinates/current/
http://orlando.drc.com/daml/Ontology/Intelligence/Report/current/
http://orlando.drc.com/daml/Ontology/POC/current/
http://orlando.drc.com/daml/ontology/UniversalProperty/current/

http://orlando.drc.com/daml/ontology/VES/current/
http://orlando.drc.com/SemanticWeb/DAML/Ontology/dc
http://orlando.drc.com/SemanticWeb/DAML/Ontology/DIS/Entity/Platform/Land
http://orlando.drc.com/SemanticWeb/DAML/Ontology/VES
http://purl.org/net/swn
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/ProfessionalExperienceAndEducation.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/beer1.0.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/cs1.0.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/cs1.1.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/docmnt1.0.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/general1.0.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/personal1.0.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/tseont.daml
http://www.cs.umd.edu/projects/plus/DAML/onts/univ1.0.daml
http://www.cs.umd.edu/~golbeck/daml/baseball.daml
http://www.cs.yale.edu/~dvm/daml/agent-ont.daml
http://www.cs.yale.edu/~dvm/daml/bib-ont.daml
http://www.cs.yale.edu/~dvm/daml/exp-ont.daml
http://www.cyc.com/2002/04/08/cyc.daml
http://www.cyc.com/2003/04/01/cyc
http://www.daml.ecs.soton.ac.uk/ont/currency.daml
http://www.daml.org/2000/10/daml-ont
http://www.daml.org/2000/12/daml+oil
http://www.daml.org/2001/03/daml+oil
http://www.daml.org/2001/06/expenses/amex-ont
http://www.daml.org/2001/06/expenses/check-ont
http://www.daml.org/2001/06/expenses/eecr-ont
http://www.daml.org/2001/06/expenses/trip-ont
http://www.daml.org/2001/09/countries/fips-10-4-ont
http://www.daml.org/2001/09/countries/iso-3166-ont
http://www.daml.org/2001/10/html/airport-ont
http://www.daml.org/2001/10/html/nyse-ont
http://www.daml.org/2001/10/html/zipcode-ont
http://www.daml.org/2002/02/telephone/1/areacodes-ont
http://www.daml.org/2002/03/metrics/metrics-ont
http://www.daml.org/2002/03/ranks/rank-ont
http://www.daml.org/2002/03/usnships/ship-ont
http://www.daml.org/2002/08/nasdaq/nasdaq-ont
http://www.daml.org/2002/09/milservices/milservices-ont
http://www.daml.org/2002/10/hazardous/hazardous-cargo-ont
http://www.daml.org/2002/10/sndl/unit-ont
http://www.daml.ri.cmu.edu/ont/homework/atlas-cmu.daml
http://www.daml.ri.cmu.edu/ont/homework/atlas-employment_categories.daml
http://www.daml.ri.cmu.edu/ont/homework/atlas-publications.daml
http://www.davincinetbook.com:8080/daml/rdf/homework3/projectGutenbergOnt.daml
http://www.davincinetbook.com:8080/daml/rdf/personal-info.daml
http://www.ics.mq.edu.au/~borgun/Software.html
http://www.imt.edu.pk
http://www.isi.edu/webscripter/bibtex.o.daml
http://www.isi.edu/webscripter/division.o.daml
http://www.isi.edu/webscripter/document.o.daml
http://www.isi.edu/webscripter/event.o.daml
http://www.isi.edu/webscripter/person.o.daml
http://www.isi.edu/webscripter/project.o.daml
http://www.isi.edu/webscripter/publication.o.daml
http://www.isi.edu/webscripter/snapshot.o.daml
http://www.kestrel.edu/DAML/2000/12/CAPACITY.daml
http://www.kestrel.edu/DAML/2000/12/DEMAND.daml
http://www.kestrel.edu/DAML/2000/12/instances.daml
http://www.kestrel.edu/DAML/2000/12/OPERATION.daml
http://www.kestrel.edu/DAML/2000/12/RESOURCE.daml
http://www.ksl.stanford.edu/projects/DAML/ksl-daml-instances.daml
http://www.ksl.stanford.edu/software/IW/spec/iw.daml
http://www.semanticweb.org/library/wordnet/wordnet-20000620.rdfs
http://www.semanticweb.org/ontologies/swrc-onto-2000-09-10.daml
http://www.tridedalo.com.br/2003/07/cns/
http://www.w3.org/2000/10/swap/infoset/infoset-diagram.rdf
http://www.w3.org/2001/03/earl/0.95.rdf
http://www.w3.org/2001/03/thread
http://www.w3.org/2001/03swell/rcs

Not OWL DL

http://opencyc.sourceforge.net/daml/naics
http://orlando.drc.com/daml/Ontology/Genealogy/current/

```
http://orlando.drc.com/daml/ontology/Glossary/current/
http://orlando.drc.com/daml/ontology/Person/current/
http://orlando.drc.com/SemanticWeb/OWL/Ontology/spaceshuttle/crew
http://orlando.drc.com/SemanticWeb/OWL/Ontology/spaceshuttle/mission
http://orlando.drc.com/semanticweb/owl/ontology/spaceshuttle/system
http://reliant.teknowledge.com/DAML/terroristAttackTypes.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Awards.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Bio.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/ContactInfo.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Course.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Date.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Image.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Organization.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Person.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Project.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Publication.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Researcher.daml
http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Topic.daml
http://www.civil.auc.dk/~i6ycl/itcode/test/itcode-projectteam.rdf
http://www.cs.yale.edu/~dvm/daml/pddlonto.daml
http://www.site.uottawa.ca/~mkhedr/ContextFinal2.daml
```

D. Statistical Analysis of DAML+OIL ontologies

This appendix describes the statistical analysis of 95 DAML+ORG ontologies, that was carried out in July 2003 in context of the (Tempich & Volz, 2003) paper.

D.1. Counts

D.1.1. Absolute Counts

Ontology	Class Desc.	Named Class	Enum.	Comple ment	Conjun ction	Disjunc tion	Restr. (Sum)	Exist. Restr.	Value Restr.	Number Restr.	Number Restr. (Lite)	Prop.	Data. Prop.	Obj. Prop.	Trans. Prop.	Inv. Funct. Prop.	Funct. Prop.	Inverse
abstract-review-o.daml	9	1	1	0	0	0	7	0	0	7	7	7	6	1	0	0	0	0
acl.daml	25	24	1	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0
acldaml.daml	10	9	1	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0
agenda-ont.daml	29	8	0	0	0	0	21	0	16	5	5	14	11	3	0	0	0	0
agent-ont2.daml	75	24	0	0	0	0	51	0	36	15	15	25	14	11	0	1	0	0
airport-ont.daml	8	1	0	0	0	0	7	0	7	0	0	7	7	0	0	0	0	0
amex-ont.daml	6	1	0	0	0	0	5	0	5	5	5	5	5	0	0	0	0	0
annotation-ns.daml	1	1	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0
areacodes-ont.daml	23	6	0	0	0	0	17	0	16	1	1	11	6	5	0	0	0	0
ArtOntology.daml	21	9	2	0	0	0	10	0	0	9	9	8	2	6	0	0	0	0
Bibliographic-ont.daml	4	4	0	0	0	0	0	0	0	0	0	66	63	3	0	0	0	0
BriefingOntology.daml	15	3	0	0	0	0	12	0	3	9	9	13	0	0	0	0	8	0
BriefingsOntology.daml	26	3	0	0	0	0	23	0	1	1	1	23	0	0	0	0	0	0
CALL-Thesaurus-ont.daml	32	6	0	0	0	0	26	0	26	15	15	40	28	12	3	0	0	8
cerif.daml	380	152	0	0	3	0	225	146	59	44	44	269	0	269	3	0	0	1
check-ont.daml	10	2	0	0	0	0	8	0	8	7	7	7	7	0	0	0	0	0
chiefs-ont.daml	6	1	0	0	0	0	5	0	3	2	2	3	1	2	0	0	0	0
CinemaAndMovies.daml	20	2	2	0	0	0	16	0	16	0	0	16	13	3	0	0	0	0
classification-ont.daml	9	3	0	0	0	0	6	0	4	2	2	4	0	4	1	0	0	0
CommercialShip-ont.daml	50	25	0	0	0	0	25	0	25	22	22	39	35	4	0	0	0	0
Condition-ont.daml	657	330	0	0	0	0	327	0	327	326	326	37	32	5	2	0	0	1
cvslog-ont.daml	21	6	0	0	0	0	15	0	8	7	7	6	3	3	0	0	0	0
daml+oil.daml	15	12	0	2	0	1	0	0	0	0	0	38	0	0	0	0	0	0
daml-ext-ont.daml	21	5	0	0	0	0	16	0	16	15	15	29	27	2	0	0	0	0
darpadir-ont.daml	9	2	0	0	0	0	7	0	4	3	3	4	3	1	0	0	0	0
datastore-schema.daml	1	1	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0
drc-ves-ont.daml	29	3	0	0	0	0	26	0	14	12	12	13	11	2	0	0	0	0
drc-ves-ont.daml.daml	29	3	0	0	0	0	26	0	14	12	12	13	11	2	0	0	0	0
DynamicOntology1.daml	3	2	1	0	0	0	0	0	0	0	0	11	0	0	0	0	1	0
eecr-ont.daml	10	2	0	0	0	0	8	0	8	8	8	7	6	1	0	0	0	0
enpmap.daml	3	1	0	0	0	0	2	0	1	1	1	1	1	0	0	0	0	0
extraction-ont.daml	107	79	0	2	0	1	25	0	13	12	11	51	0	13	0	0	0	6
factbook-ont.daml	193	35	0	0	0	0	158	0	158	0	0	171	145	26	0	0	0	0
fips-10-4-ont.daml	3	1	0	0	0	0	2	0	2	2	2	2	2	0	0	0	0	0
Fugitive-ont.daml	8	8	0	0	0	0	0	0	0	0	0	48	44	4	0	0	0	0
Gentology-ont.daml	17	13	0	0	0	0	4	0	0	2	2	43	30	11	0	0	2	0
geonames-ont.daml	14	2	0	0	0	0	12	0	12	0	0	13	9	4	0	0	0	0
Glossary-ont.daml	8	7	0	0	0	0	1	0	0	1	1	147	108	39	0	0	0	34
GoalsAndObjectives-ont.daml	50	20	0	0	0	0	30	0	18	12	12	14	11	3	0	0	0	0
GPS-ont.daml	67	18	0	0	0	0	49	0	23	26	26	15	12	3	0	0	0	0
hazardous-cargo-ont.daml	15	3	0	0	0	0	12	0	12	0	0	12	9	3	0	0	0	0
ImageFingerprintingOntology-web.daml	6	3	0	0	0	0	3	0	0	1	1	4	0	0	0	0	0	0
ImageFingerprintsOntology-briefings.daml	47	17	0	2	0	1	27	0	1	3	3	65	0	0	0	0	0	0
Imaging.daml	4	2	0	0	0	0	2	0	0	2	2	3	2	1	0	0	0	0
Intel-Report-ont.daml	20	20	0	0	0	0	0	0	0	0	0	80	54	26	0	0	0	0
iso-3166-ont.daml	3	1	0	0	0	0	2	0	2	2	2	2	2	0	0	0	0	0
Itinerary-ont.daml	34	5	3	0	0	0	26	0	26	20	20	22	15	7	0	0	0	0
Itinerary-ont2.daml	34	5	3	0	0	0	26	0	26	20	20	22	15	7	0	0	0	0
Ibi-ont.daml	3	3	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
ksl-daml-desc.daml	23	23	0	0	0	0	0	0	0	0	0	41	0	0	0	0	0	0
ksl-daml-instances.daml	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
land-platform-ont.daml	888	888	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Locator-ont.daml	44	17	0	0	0	0	27	0	27	26	26	48	46	2	0	0	0	0
map-ont.daml	37	10	1	0	0	0	26	0	13	13	12	10	5	5	0	0	0	0
moda.daml	10	3	0	0	0	0	7	0	4	3	3	3	1	2	0	0	0	0
moda-ont.daml	21	9	0	0	0	0	12	2	7	5	5	11	5	6	0	0	7	0
metrics-ont.daml	8	2	0	0	0	0	6	0	6	0	0	5	4	1	0	0	0	0
milservices-ont.daml	5	1	0	0	0	0	4	0	4	0	0	6	3	3	1	0	0	0
movenight-ont.daml	13	6	0	0	0	0	7	0	0	7	7	10	3	7	0	0	0	0
nasdaq-ont.daml	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
NSO-ont.daml	86	42	0	0	0	0	44	6	16	22	15	24	11	13	2	0	0	1
nyse-ont.daml	17	1	0	0	0	0	16	0	8	8	8	9	7	2	0	1	0	0
office.daml	19	19	0	0	0	0	0	0	0	0	0	14	2	12	0	0	0	0
ontologies-ont.daml	24	4	0	0	0	0	20	0	20	16	16	19	17	2	0	0	0	0
Ontology.daml	27	27	0	0	0	0	0	0	0	0	0	20	0	0	0	0	14	0
Organization-ont.daml	5	5	0	0	0	0	0	0	0	0	0	41	34	7	2	0	0	1
Person-ont.daml	3	3	0	0	0	0	0	0	0	0	0	42	39	3	0	0	0	0
POC-ont.daml	51	24	0	0	0	0	27	0	27	26	26	105	90	15	2	0	0	1
pptOntology.daml	3	2	0	0	0	0	1	0	0	0	0	18	17	1	0	0	0	0
projectGutenbergOnt.daml	22	5	0	0	0	0	17	0	17	13	13	10	5	5	0	0	0	0
projectplan.daml	36	19	0	0	0	0	17	0	0	0	0	13	0	0	0	0	0	0
projects-20010811.daml	49	15	0	0	0	0	34	0	34	0	0	26	12	14	0	0	0	0
rank-ont.daml	6	3	0	0	0	0	3	0	3	0	0	5	2	3	1	0	0	0
running.daml	10	10	0	0	0	0	0	0	0	0	0	54	0	0	0	0	0	0
scientific-review-o.daml	3	1	0	0	0	0	2	0	0	2	2	2	2	0	0	0	0	0
ship-ont.daml	7	2	0	0	0	0	5	0	5	0	0	5	4	1	0	0	0	0
shoeproj-ont.daml	24	18	0	2	0	1	3	0	3	0	0	41	0	3	0	0	0	0
SUMO.daml	583	583	0	0	0	0	0	0	0	0	0	150	6	144	0	0	0	0
SurveyOntology.daml	40	15	0	0	0	6	19	0	10	16	16	14	0	0	0	0	3	0
tambis.daml	540	540	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0
TaskList-ont.daml	35	7	0	0	0	0	28	0	28	21	21	42	34	8	2	0	0	1
TaskListUUTLScenario-ont.daml	56	13	0	0	0	0	43	0	42	27	27	56	35	21	2	0	0	1
terroristAttackTypes.daml	66	66	0	0	0	0	0	0	0	0	0	8	6	2	0	0	0	0
Terrorists.daml	2	2	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
tools-ont.daml	16	4	0	0	0	0	12	0	12	2	2	11	9	2	0	0	0	0
ttp-ont.daml	11	2	0	0	0	0	9	0	9	5	5	9	5	4	0	0	0	0
unit-ont.daml	7	1	0	0	0	0	6	0	4	2	2	3	1	2	0	0	0	0
units-ont.daml	10	5	0	0	0	0	5	0	5	0	0	5	2	3	0	0	0	0
UniversalProperty-ont.daml	2	2	0	0	0	0	0	0	0	0	0	29	29	0	0	0	0	0
UNSPSC.daml	9795	9795	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
vegetarian.daml	32	19	0	4	4	0	5	0	5	0	0	1	0	0	0	0	0	0
wines.daml	258	56	16	0	78	0	118	33	32	0	0	10	0	0	0	0	8	0
WMD.daml	162	162	0	0	0	0	0	0	0	0	0	9	1	8	0	0	0	0
world-fact-book.daml	1369	1327	0	0	0	31	11	0	5	0	0	179	0	3	0	0	10	1
zipcode-ont.daml	**15**	**2**	**0**	**0**	**0**	**0**	**13**	**0**	**7**	**6**	**6**	**7**	**5**	**2**	**0**	**0**	**0**	**0**

D.1.2. Relative Counts

Ontology	Named / Class	Obj. Prop./ Prop.	Dat. Prop. / Prop.	Untyped / Prop	Prop. / Named	Dat. Prop. / Named	Obj. Prop. / Named	Descr.	Dat. Prop. / Class	Obj. Prop. / Class	Trans. / Obj. Prop.	Exist. / Restr.
abstract-review-o.daml	13%	29%	71%	0%	3,50	2,50	1,00	0,47	0,13	0,33	0%	0%
acl.daml	97%	2%	0%	98%	0,13	0,00	0,00	0,13	0,00	0,00	0%	0%
acldaml.daml	100%	89%	11%	0%	0,06	0,01	0,05	0,06	0,05	0,01	0%	0%
agenda-ont.daml	21%	0%	0%	100%	0,18	0,00	0,00	0,04	0,00	0,00	0%	28%
agent-ont2.daml	59%	0%	0%	100%	0,05	0,00	0,00	0,03	0,00	0,00	0%	0%
airport-ont.daml	100%	0%	100%	0%	0,00	0,00	0,00	0,00	0,00	0,00	0%	0%
amex-ont.daml	100%	0%	100%	0%	14,50	14,50	0,00	14,50	0,00	14,50	0%	0%
annotation-ns.daml	50%	60%	40%	0%	1,00	0,40	0,60	0,50	0,30	0,20	0%	0%
areacodes-ont.daml	14%	67%	33%	0%	3,00	1,00	2,00	0,43	0,29	0,14	0%	0%
ArtOntology.daml	18%	44%	56%	0%	4,50	2,50	2,00	0,82	0,36	0,45	0%	0%
Bibliographic-ont.daml	25%	18%	82%	0%	2,75	2,25	0,50	0,69	0,13	0,56	0%	0%
BriefingOntology.daml	100%	0%	100%	0%	0,50	0,50	0,00	0,50	0,00	0,50	0%	0%
BriefingsOntology.daml	100%	25%	75%	0%	0,12	0,09	0,03	0,12	0,03	0,09	0%	0%
CALL-Thesaurus-ont.daml	23%	38%	63%	0%	4,31	2,69	1,62	1,00	0,38	0,63	10%	0%
cerif.daml	20%	19%	81%	0%	6,00	4,86	1,14	1,20	0,23	0,97	25%	0%
check-ont.daml	100%	0%	0%	100%	0,19	0,00	0,00	0,19	0,00	0,00	0%	0%
chiefs-ont.daml	38%	0%	0%	100%	0,93	0,00	0,00	0,35	0,00	0,00	0%	0%
CinemaAndMovies.daml	100%	96%	4%	0%	0,26	0,01	0,25	0,26	0,25	0,01	0%	0%
classification-ont.daml	75%	7%	0%	93%	2,28	0,00	0,17	1,71	0,13	0,00	0%	0%
CommercialShip-ont.daml	29%	20%	80%	0%	2,50	2,00	0,50	0,71	0,14	0,57	0%	0%
Condition-ont.daml	33%	0%	100%	0%	2,00	2,00	0,00	0,67	0,00	0,67	0%	0%
cvslog-ont.daml	100%	0%	0%	100%	5,40	0,00	0,00	5,40	0,00	0,00	0%	0%
daml+oil.daml	50%	60%	40%	0%	1,67	0,67	1,00	0,83	0,50	0,33	33%	0%
daml-ext-ont.daml	31%	54%	46%	0%	1,73	0,80	0,93	0,53	0,29	0,24	0%	0%
darpadir-ont.daml	53%	0%	0%	100%	0,68	0,00	0,00	0,36	0,00	0,00	0%	0%
datastore-schema.daml	23%	50%	50%	0%	2,00	1,00	1,00	0,45	0,23	0,23	0%	0%
drc-ves-ont.daml	67%	6%	94%	0%	9,00	8,50	0,50	6,00	0,33	5,67	0%	0%
drc-ves-ont.daml.daml	47%	14%	86%	0%	4,38	3,75	0,63	2,06	0,29	1,76	13%	0%
DynamicOntology1.daml	100%	7%	93%	0%	14,00	13,00	1,00	14,00	1,00	13,00	0%	0%
eecr-ont.daml	100%	17%	83%	0%	8,20	6,80	1,40	8,20	1,40	6,80	29%	0%
enpmap.daml	100%	0%	0%	100%	0,74	0,00	0,00	0,74	0,00	0,00	0%	0%
extraction-ont.daml	17%	11%	89%	0%	4,75	4,25	0,50	0,79	0,08	0,71	0%	0%
factbook-ont.daml	100%	86%	14%	0%	0,74	0,11	0,63	0,74	0,63	0,11	0%	0%
fips-10-4-ont.daml	6%	22%	78%	0%	9,00	7,00	2,00	0,53	0,12	0,41	0%	0%
Fugitive-ont.daml	49%	54%	46%	0%	0,57	0,26	0,31	0,28	0,15	0,13	15%	14%
Gentology-ont.daml	100%				0,00	0,00	0,00	0,00	0,00	0,00	0%	0%
geonames-ont.daml	46%	70%	30%	0%	1,67	0,50	1,17	0,77	0,54	0,23	0%	0%
Glossary-ont.daml	20%	50%	50%	0%	6,00	3,00	3,00	1,20	0,60	0,60	33%	0%
GoalsAndObjectives-ont.daml	25%	20%	80%	0%	2,50	2,00	0,50	0,63	0,13	0,50	0%	0%
GPS-ont.daml	43%	55%	45%	0%	1,22	0,56	0,67	0,52	0,29	0,24	0%	17%
hazardous-cargo-ont.daml	30%	67%	33%	0%	1,00	0,33	0,67	0,30	0,20	0,10	0%	0%
ImageFingerprintingOntology-web.daml	27%	50%	50%	0%	1,00	0,50	0,50	0,27	0,14	0,14	0%	0%
ImageFingerprintsOntology-briefings.daml	39%	4%	96%	0%	2,82	2,71	0,12	1,09	0,05	1,05	0%	0%
Imaging.daml	100%				0,00	0,00	0,00	0,00	0,00	0,00	0%	0%
Intel-Report-ont.daml	100%				0,00	0,00	0,00	0,00	0,00	0,00	0%	0%
iso-3166-ont.daml	100%	0%	0%	100%	1,78	0,00	0,00	1,78	0,00	0,00	0%	0%
itinerary-ont.daml	100%	0%	0%	100%	0,33	0,00	0,00	0,33	0,00	0,00	0%	0%
itinerary-ont2.daml	15%	32%	68%	0%	4,40	3,00	1,40	0,65	0,21	0,44	0%	0%
jbi-ont.daml	15%	32%	68%	0%	4,40	3,00	1,40	0,65	0,21	0,44	0%	0%
ksl-daml-desc.daml	33%	0%	100%	0%	2,00	2,00	0,00	0,67	0,00	0,67	0%	0%
ksl-daml-instances.daml	100%				4,00	2,70	1,30	4,00	1,30	2,70	0%	0%
land-platform-ont.daml	50%				1,50	1,00	0,50	0,75	0,25	0,50	0%	0%
Locator-ont.daml	36%	0%	0%	100%	3,82	0,00	0,00	1,38	0,00	0,00	0%	0%
map-ont.daml	50%	0%	0%	100%	1,33	0,00	0,00	0,67	0,00	0,00	0%	0%
mcda.daml	20%	25%	75%	0%	4,00	3,00	1,00	0,80	0,20	0,60	0%	0%
mcda-ont.daml	27%	20%	80%	0%	0,83	0,67	0,17	0,22	0,04	0,18	0%	0%
metrics-ont.daml	40%	21%	79%	0%	0,70	0,55	0,15	0,28	0,06	0,22	0%	0%
milservices-ont.daml	88%	27%	73%	0%	21,00	15,43	5,57	18,38	4,88	13,50	0%	0%
movienight-ont.daml	14%	31%	69%	0%	6,50	4,50	2,00	0,93	0,29	0,64	0%	0%
nasdaq-ont.daml	76%				3,31	2,31	0,85	2,53	0,65	1,76	0%	0%
NSO-ont.daml	100%	8%	92%	0%	6,00	5,50	0,50	6,00	0,50	5,50	0%	0%
nyse-ont.daml	33%	0%	100%	0%	2,00	2,00	0,00	0,67	0,00	0,67	0%	0%
office.daml	18%	15%	85%	0%	4,89	4,14	0,74	0,89	0,13	0,75	0%	0%
ontologies-ont.daml	74%	25%	0%	75%	0,65	0,00	0,16	0,48	0,12	0,00	0%	0%
Ontology.daml	33%	0%	100%	0%	1,00	1,00	0,00	0,33	0,00	0,33	0%	0%
Organization-ont.daml	20%	14%	86%	0%	3,50	3,00	0,50	0,70	0,10	0,60	0%	0%
Person-ont.daml	67%	0%	0%	100%	5,50	0,00	0,00	3,67	0,00	0,00	0%	0%
POC-ont.daml	10%	15%	85%	0%	4,33	3,67	0,67	0,45	0,07	0,38	0%	0%
pptOntology.daml	10%	15%	85%	0%	4,33	3,67	0,67	0,45	0,07	0,38	0%	0%
projectGutenbergOnt.daml	100%	0%	0%	100%	16,00	0,00	0,00	16,00	0,00	0,00	0%	0%
projectplan.daml	22%	25%	75%	0%	2,00	1,50	0,50	0,44	0,11	0,33	0%	0%
projects-20010811.daml	24%	7%	93%	0%	5,80	5,40	0,40	1,38	0,10	1,29	0%	0%
rank-ont.daml	80%	0%	0%	100%	3,17	0,00	0,00	2,53	0,00	0,00	0%	0%
running.daml	29%	50%	50%	0%	1,00	0,50	0,50	0,29	0,14	0,14	0%	0%
scientific-review-o.daml	50%	14%	86%	0%	0,11	0,10	0,02	0,06	0,01	0,05	40%	0%
ship-ont.daml	50%	10%	90%	0%	1,56	1,40	0,16	0,78	0,08	0,70	0%	0%
shoeproj-ont.daml	33%	100%	0%	0%	1,33	0,00	1,33	0,44	0,44	0,00	25%	0%
SUMO.daml	10%	19%	81%	0%	8,00	6,50	1,50	0,80	0,15	0,65	0%	0%
SurveyOntology.daml	17%	67%	33%	0%	3,00	1,00	2,00	0,50	0,33	0,17	0%	0%
tambis.daml	20%	0%	100%	0%	3,50	3,50	0,00	0,70	0,00	0,70	0%	0%
TaskList-ont.daml	40%	100%	0%	0%	1,77	0,00	1,77	0,71	0,71	0,00	1%	65%
TaskListUJTLScenario-ont.daml	19%	30%	70%	0%	6,67	4,67	2,00	1,25	0,38	0,88	25%	0%
terroristAttackTypes.daml	12%	0%	0%	100%	7,67	0,00	0,00	0,88	0,00	0,00	0%	0%
Terrorists.daml	20%	0%	0%	100%	4,33	0,00	0,00	0,87	0,00	0,00	0%	0%
tools-ont.daml	100%	5%	95%	0%	16,50	15,75	0,75	16,50	0,75	15,75	0%	0%
trip-ont.daml	43%	75%	25%	0%	0,89	0,22	0,67	0,38	0,29	0,10	0%	0%
unit-ont.daml	26%	45%	55%	0%	1,83	1,00	0,83	0,48	0,22	0,26	0%	0%
units-ont.daml	100%	0%	0%	100%	7,00	0,00	0,00	7,00	0,00	0,00	0%	0%
UniversalProperty-ont.daml	17%	0%	100%	0%	5,00	5,00	0,00	0,83	0,00	0,83	0%	0%
UNSPSC.daml	13%	0%	100%	0%	7,00	7,00	0,00	0,88	0,00	0,88	0%	0%
vegetarian.daml	32%	44%	56%	0%	1,04	0,58	0,46	0,33	0,15	0,19	0%	0%
wines.daml	28%	21%	79%	0%	1,75	1,38	0,38	0,48	0,10	0,38	0%	0%
WMD.daml	90%	0%	0%	100%	1,78	0,00	0,00	1,60	0,00	0,00	0%	0%
world-fact-book.daml	96%	0%	0%	100%	0,50	0,00	0,00	0,48	0,00	0,00	0%	0%
zipcode-ont.daml	11%	14%	86%	0%	7,00	6,00	1,00	0,78	0,67	0,11	0%	0%

D.2. Distributions

D.2.1. Super-Class Distributions

Ontology	#Class Desc.	Number of Super-Classes per Class Description															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
abstract-review-o.daml	9	88,89%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	11,11%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
acl.daml	25	4,00%	96,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
acldaml.daml	10	10,00%	90,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
agenda-ont.daml	29	75,86%	0,00%	10,34%	3,45%	3,45%	3,45%	0,00%	0,00%	3,45%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
agent-ont2.daml	75	73,33%	9,33%	6,67%	4,00%	1,33%	0,00%	0,00%	0,00%	1,33%	2,67%	1,33%	0,00%	0,00%	0,00%	0,00%	
airport-ont.daml	8	87,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	12,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
annex-ont.daml	6	83,33%	0,00%	0,00%	0,00%	0,00%	16,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
annotation-ns.daml	1	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
areacodes-ont.daml	23	73,91%	8,70%	8,70%	0,00%	4,35%	0,00%	0,00%	4,35%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ArtOntology.daml	21	57,14%	28,57%	9,52%	0,00%	0,00%	4,76%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Bibliographic-ont.daml	4	50,00%	50,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
BriefingOntology.daml	15	80,00%	0,00%	6,67%	6,67%	0,00%	0,00%	0,00%	6,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
BriefingsOntology.daml	26	92,31%	3,85%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	3,85%	
CALL-Thesaurus-ont.daml	32	81,25%	6,25%	3,13%	3,13%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	3,13%	3,13%	0,00%	0,00%	0,00%	
cerif.daml	380	67,37%	16,84%	5,79%	3,68%	2,37%	1,32%	0,26%	0,53%	0,26%	0,53%	0,00%	0,26%	0,26%	0,53%	0,00%	
check-ont.daml	10	80,00%	10,00%	0,00%	0,00%	0,00%	0,00%	0,00%	10,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
chiefs-ont.daml	6	83,33%	0,00%	0,00%	0,00%	16,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
CinemaAndMovies.daml	20	90,00%	0,00%	0,00%	0,00%	0,00%	5,00%	0,00%	0,00%	0,00%	0,00%	0,00%	5,00%	0,00%	0,00%	0,00%	
classification-ont.daml	9	77,78%	0,00%	11,11%	11,11%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
CommercialShip-ont.daml	50	50,00%	42,00%	2,00%	2,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	4,00%	0,00%	0,00%	0,00%	0,00%	
Condition-ont.daml	657	49,92%	2,13%	47,64%	0,15%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,15%	0,00%	0,00%	0,00%	0,00%	
cvslog-ont.daml	21	71,43%	9,52%	4,76%	4,76%	4,76%	0,00%	4,76%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
daml+oil.daml	15	40,00%	60,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
daml-ext-ont.daml	21	76,19%	9,52%	0,00%	9,52%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	4,76%	0,00%	0,00%	0,00%	0,00%	
darpadir-ont.daml	9	77,78%	0,00%	0,00%	0,00%	11,11%	11,11%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
datastore-schema.daml	1	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
drc-ves-ont.daml	29	89,66%	0,00%	6,90%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	3,45%	
drc-ves-ont.daml.daml	29	89,66%	0,00%	6,90%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	3,45%	
DynamicOntology1.daml	3	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
eecr-ont.daml	10	80,00%	0,00%	0,00%	10,00%	0,00%	10,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
enpmap.daml	3	66,67%	0,00%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
extraction-ont.daml	107	32,71%	61,68%	0,93%	1,87%	0,00%	0,00%	0,93%	0,93%	0,00%	0,00%	0,93%	0,00%	0,00%	0,00%	0,00%	
factbook-ont.daml	193	82,38%	9,33%	7,25%	0,52%	0,52%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
fips-10-4-ont.daml	3	66,67%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Fugitive-ont.daml	8	25,00%	75,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Gentology-ont.daml	17	29,41%	52,94%	17,65%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
geonames-ont.daml	14	85,71%	0,00%	0,00%	7,14%	0,00%	0,00%	0,00%	0,00%	7,14%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Glossary-ont.daml	8	62,50%	37,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
GoalsAndObjectives-ont.daml	50	64,00%	2,00%	28,00%	4,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	2,00%	
GPS-ont.daml	67	73,13%	0,00%	11,94%	0,00%	4,48%	7,46%	1,49%	0,00%	0,00%	1,49%	0,00%	0,00%	0,00%	0,00%	0,00%	
hazardous-cargo-ont.daml	15	80,00%	6,67%	0,00%	6,67%	0,00%	0,00%	0,00%	0,00%	0,00%	6,67%	0,00%	0,00%	0,00%	0,00%	0,00%	
ImageFingerprintingOntology-web.daml	6	50,00%	50,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ImageFingerprintsOntology-briefings.daml	47	72,34%	23,40%	2,13%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	2,13%	
Imaging.daml	4	50,00%	50,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Intel-Report-ont.daml	20	65,00%	35,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
iso-3166-ont.daml	3	66,67%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Itinerary-ont.daml	34	88,24%	0,00%	2,94%	0,00%	0,00%	2,94%	0,00%	2,94%	0,00%	0,00%	0,00%	0,00%	2,94%	0,00%	0,00%	
Itinerary-ont2.daml	34	88,24%	0,00%	2,94%	0,00%	0,00%	2,94%	0,00%	2,94%	0,00%	0,00%	0,00%	0,00%	2,94%	0,00%	0,00%	
lbi-ont.daml	3	0,00%	33,33%	0,00%	0,00%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	
ksl-daml-desc.daml	23	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ksl-daml-instances.daml	1	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
land-platform-ont.daml	888	1,01%	98,20%	0,79%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Locator-ont.daml	44	63,64%	13,64%	15,91%	2,27%	0,00%	0,00%	0,00%	2,27%	0,00%	0,00%	2,27%	0,00%	0,00%	0,00%	0,00%	
map-ont.daml	37	72,97%	8,11%	5,41%	0,00%	5,41%	5,41%	0,00%	2,70%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
moda.daml	10	70,00%	10,00%	0,00%	10,00%	0,00%	0,00%	10,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
moda-ont.daml	21	61,90%	9,52%	9,52%	9,52%	9,52%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
metrics-ont.daml	8	75,00%	0,00%	12,50%	0,00%	12,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
milservices-ont.daml	5	80,00%	0,00%	0,00%	0,00%	20,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
movenight-ont.daml	13	69,23%	7,69%	15,38%	7,69%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
nasdaq-ont.daml	3	66,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	
NSO-ont.daml	86	55,81%	31,40%	3,49%	5,81%	1,16%	0,00%	1,16%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	1,16%	
nyse-ont.daml	17	94,12%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	5,88%	
office.daml	19	78,95%	21,05%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ontologies-ont.daml	24	83,33%	4,17%	0,00%	8,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	4,17%	
Ontology.daml	27	25,93%	62,96%	11,11%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Organization-ont.daml	5	40,00%	60,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Person-ont.daml	3	33,33%	66,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
POC-ont.daml	51	62,75%	17,65%	13,73%	1,96%	0,00%	0,00%	0,00%	1,96%	0,00%	0,00%	1,96%	0,00%	0,00%	0,00%	0,00%	
pptOntology.daml	3	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
projectGutenbergOnt.daml	22	77,27%	0,00%	18,18%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	4,55%	0,00%	0,00%	0,00%	0,00%	0,00%	
projectplan.daml	36	75,00%	11,11%	8,33%	0,00%	0,00%	2,78%	2,78%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
projects-20010811.daml	49	71,43%	0,00%	6,12%	10,20%	8,16%	2,04%	0,00%	2,04%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
rank-ont.daml	6	83,33%	0,00%	0,00%	16,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
running.daml	10	90,00%	10,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
scientific-review-o.daml	3	66,67%	0,00%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ship-ont.daml	7	85,71%	0,00%	0,00%	0,00%	0,00%	14,29%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
shoeproj-ont.daml	24	41,67%	45,83%	8,33%	0,00%	4,17%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
SUMO.daml	583	0,34%	89,71%	9,43%	0,51%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
SurveyOntology.daml	40	50,00%	20,00%	22,50%	5,00%	0,00%	0,00%	2,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
tambis.daml	540	53,33%	39,44%	2,22%	2,96%	0,74%	0,74%	0,00%	0,37%	0,00%	0,00%	0,19%	0,00%	0,00%	0,00%	0,00%	
TaskList-ont.daml	35	80,00%	2,86%	2,86%	2,86%	2,86%	0,00%	2,86%	0,00%	0,00%	0,00%	5,71%	0,00%	0,00%	0,00%	0,00%	
TaskListUJTLScenario-ont.daml	56	76,79%	1,79%	1,79%	5,36%	1,79%	0,00%	1,79%	7,14%	0,00%	0,00%	3,57%	0,00%	0,00%	0,00%	0,00%	
terroristAttackTypes.daml	66	1,52%	93,94%	4,55%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Terrorists.daml	2	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
tools-ont.daml	16	75,00%	6,25%	12,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	6,25%	0,00%	0,00%	0,00%	0,00%	0,00%	
trip-ont.daml	11	81,82%	0,00%	9,09%	0,00%	0,00%	0,00%	0,00%	9,09%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
unit-ont.daml	7	85,71%	0,00%	0,00%	0,00%	0,00%	0,00%	14,29%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
units-ont.daml	10	60,00%	10,00%	20,00%	10,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
UniversalProperty-ont.daml	2	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
UNSPSC.daml	9795	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
vegetarian.daml	32	62,50%	12,50%	21,88%	3,13%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
wines.daml	268	57,84%	24,63%	4,48%	2,99%	7,09%	1,87%	0,37%	0,37%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,37%	
WMD.daml	162	0,00%	82,72%	14,20%	3,09%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
world-fact-book.daml	1369	23,81%	70,49%	5,41%	0,22%	0,07%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
zipcode-ont.daml	15	86,67%	0,00%	0,00%	0,00%	0,00%	6,67%	0,00%	0,00%	6,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	

D.2.2. Restriction Distributions

		Number of Restrictions per Class Description															
Ontology	#Restr.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
abstract-review-o.daml	7	88,89%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	11,11%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
acl.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
acldaml.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
agenda-ont.daml	21	75,86%	10,34%	3,45%	3,45%	0,00%	3,45%	0,00%	0,00%	3,45%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
agent-ont2.daml	51	82,67%	4,00%	6,67%	0,00%	1,33%	0,00%	0,00%	1,33%	1,33%	1,33%	1,33%	0,00%	0,00%	0,00%	0,00%	
airport-ont.daml	7	87,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	12,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
amex-ont.daml	5	83,33%	0,00%	0,00%	0,00%	0,00%	16,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
annotation-ns.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
areacodes-ont.daml	17	73,91%	8,70%	8,70%	0,00%	4,35%	0,00%	0,00%	4,35%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ArtOntology.daml	10	80,95%	4,76%	9,52%	0,00%	0,00%	4,76%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Bibliographic-ont.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
BriefingOntology.daml	12	80,00%	0,00%	6,67%	6,67%	0,00%	0,00%	0,00%	6,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
BriefingsOntology.daml	23	92,31%	3,85%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	3,85%	
CALL-Thesaurus-ont.daml	26	87,50%	0,00%	3,13%	3,13%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	3,13%	3,13%	0,00%	0,00%	0,00%	
cerif.daml	225	85,53%	0,79%	3,95%	4,74%	1,05%	1,32%	0,26%	0,53%	0,26%	0,53%	0,00%	0,26%	0,26%	0,53%	0,00%	
check-ont.daml	8	80,00%	10,00%	0,00%	0,00%	0,00%	0,00%	0,00%	10,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
chiefs-ont.daml	5	83,33%	0,00%	0,00%	0,00%	16,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
CinemaAndMovies.daml	16	90,00%	0,00%	0,00%	0,00%	0,00%	5,00%	0,00%	0,00%	0,00%	0,00%	0,00%	5,00%	0,00%	0,00%	0,00%	
classification-ont.daml	6	77,78%	0,00%	22,22%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
CommercialShip-ont.daml	25	92,00%	0,00%	2,00%	2,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	4,00%	0,00%	0,00%	0,00%	0,00%	
Condition-ont.daml	327	52,05%	47,49%	0,15%	0,15%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,15%	0,00%	0,00%	0,00%	0,00%	
cvslog-ont.daml	15	80,95%	0,00%	4,76%	4,76%	4,76%	0,00%	4,76%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
daml+oil.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
daml-ext-ont.daml	16	85,71%	0,00%	0,00%	9,52%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	4,76%	0,00%	0,00%	0,00%	0,00%	
darpa-dir-ont.daml	7	77,78%	0,00%	0,00%	11,11%	11,11%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
datastore-schema.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
dtc-ves-ont.daml	26	89,66%	0,00%	6,90%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	3,45%	
dtc-ves-ont.daml.daml	26	89,66%	0,00%	6,90%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	3,45%	
DynamicOntology1.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
eecr-ont.daml	8	80,00%	0,00%	0,00%	10,00%	0,00%	10,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
enprmap.daml	2	66,67%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
extraction-ont.daml	25	94,39%	0,93%	1,87%	0,00%	0,00%	0,93%	0,93%	0,00%	0,00%	0,93%	0,00%	0,00%	0,00%	0,00%	0,00%	
factbook-ont.daml	158	89,64%	7,25%	2,59%	0,52%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
fips-10-4-ont.daml	2	66,67%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Fugitive-ont.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Gentology-ont.daml	4	82,35%	11,76%	5,88%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
geonames-ont.daml	12	85,71%	0,00%	0,00%	7,14%	0,00%	0,00%	0,00%	0,00%	7,14%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Glossary-ont.daml	1	87,50%	12,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
GoalsAndObjectives-ont.daml	30	86,00%	8,00%	4,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	2,00%	
GPS-ont.daml	49	82,09%	0,00%	2,99%	1,49%	10,45%	0,00%	1,49%	0,00%	1,49%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
hazardous-cargo-ont.daml	12	80,00%	6,67%	6,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	6,67%	0,00%	0,00%	0,00%	0,00%	0,00%	
ImageFingerprintingOntology-web.daml	3	50,00%	50,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ImageFingerprintsOntology-briefings.daml	27	91,49%	4,26%	2,13%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	2,13%	
Imaging.daml	2	50,00%	50,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Intel-Report-ont.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
iso-3166-ont.daml	2	66,67%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Itinerary-ont.daml	26	88,24%	0,00%	2,94%	0,00%	0,00%	2,94%	0,00%	2,94%	0,00%	0,00%	0,00%	0,00%	2,94%	0,00%	0,00%	
Itinerary-ont2.daml	26	88,24%	0,00%	2,94%	0,00%	0,00%	2,94%	0,00%	2,94%	0,00%	0,00%	0,00%	0,00%	2,94%	0,00%	0,00%	
kb-ont.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ksl-daml-desc.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ksl-daml-instances.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
land-platform-ont.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Locator-ont.daml	27	77,27%	13,64%	2,27%	2,27%	0,00%	0,00%	2,27%	0,00%	0,00%	0,00%	2,27%	0,00%	0,00%	0,00%	0,00%	
map-ont.daml	26	78,38%	5,41%	2,70%	0,00%	10,81%	0,00%	2,70%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
mcda.daml	7	80,00%	0,00%	10,00%	0,00%	0,00%	10,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
mcda-ont.daml	12	71,43%	9,52%	9,52%	9,52%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
metrics-ont.daml	6	75,00%	0,00%	12,50%	0,00%	12,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
milservices-ont.daml	4	80,00%	0,00%	0,00%	0,00%	20,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
mowenlght-ont.daml	7	69,23%	15,38%	7,69%	7,69%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
nasdaq-ont.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
NSO-ont.daml	44	88,37%	0,00%	8,14%	1,16%	0,00%	1,16%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	1,16%	
nyse-ont.daml	16	94,12%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	5,88%	
office.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ontologies-ont.daml	20	83,33%	4,17%	0,00%	8,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	4,17%	
Ontology.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Organization-ont.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Person-ont.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
POC-ont.daml	27	80,39%	11,76%	1,96%	1,96%	0,00%	0,00%	1,96%	0,00%	0,00%	0,00%	1,96%	0,00%	0,00%	0,00%	0,00%	
pptOntology.daml	1	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
projectGutenbergOnt.daml	17	77,27%	0,00%	18,18%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	4,55%	0,00%	0,00%	0,00%	0,00%	0,00%	
projectplan.daml	17	86,11%	0,00%	8,33%	0,00%	0,00%	2,78%	2,78%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
projects-2001011.daml	34	77,55%	2,04%	10,20%	2,04%	4,08%	2,04%	0,00%	2,04%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
rank-ont.daml	3	83,33%	0,00%	0,00%	16,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
running.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
scientific-review-o.daml	2	66,67%	0,00%	33,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
ship-ont.daml	5	85,71%	0,00%	0,00%	0,00%	0,00%	14,29%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
shoeproj-ont.daml	3	91,67%	4,17%	4,17%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
SUMO.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
SurveyOntology.daml	19	75,00%	12,50%	10,00%	0,00%	0,00%	0,00%	2,50%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
tambis.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
TaskList-ont.daml	28	82,86%	0,00%	2,86%	2,86%	2,86%	0,00%	2,86%	0,00%	0,00%	0,00%	5,71%	0,00%	0,00%	0,00%	0,00%	
TaskListUTLScenario-ont.daml	43	78,57%	0,00%	3,57%	3,57%	1,79%	0,00%	8,93%	0,00%	0,00%	0,00%	3,57%	0,00%	0,00%	0,00%	0,00%	
terroristAttackTypes.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
Terrorists.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
tools-ont.daml	12	75,00%	18,75%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	6,25%	0,00%	0,00%	0,00%	0,00%	0,00%	
trip-ont.daml	9	81,82%	0,00%	9,09%	0,00%	0,00%	0,00%	0,00%	9,09%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
unit-ont.daml	6	85,71%	0,00%	0,00%	0,00%	0,00%	0,00%	14,29%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
units-ont.daml	5	70,00%	10,00%	20,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
UniversalProperty-ont.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
UNSPSC.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
vegetarian.daml	5	84,38%	15,63%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
wines.daml	118	75,37%	7,84%	4,85%	2,24%	7,84%	0,75%	0,75%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,37%	
WMD.daml	0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
world-fact-book.daml	11	99,27%	0,66%	0,07%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	
zipcode-ont.daml	13	86,67%	0,00%	0,00%	0,00%	0,00%	6,67%	0,00%	0,00%	6,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	

Bibliography

Abiteboul, S. & Bidoit, N. (1986). Non-first normal form relations: an algebra allowing data restructuring. *Journal of Computer and System Sciences*, 33:361–393.

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., & Tolle, K. (2001). The ics-forth rdfsuite: Managing voluminous rdf description bases. In *2nd International Workshop on the Semantic Web (SemWeb'01)*, pages 1–13, Hongkong, China.

Andeka, H. & Nemeti, I. (1978). The generalized completeness of Horn predicate logic as a programming language. *Acta Cybernetica*, 4:3–10.

Apt, K. & Bol, R. (1994). Logic programming and negation: A survey. *Journal of Logic Programming*, 19/20:9–71.

Apt, K. & Pugin, J.-M. (1987). Maintenance of stratified databases viewed as belief revision system. In *Proc. of the 6th Symposium on Principles of Database Systems (PODS)*, pages 136–145, San Diego, CA, USA.

Areces, C. & Lutz, C. (2002). Concrete domains and nominals united. In *Proc. of the 4th Workshop on Hybrid Logics*.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. F. (Eds.) (2003). *The Description Logic Handbook*. Cambridge University Press.

Baader, F. & Hanschke, P. (1991). A schema for integrating concrete domains into concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457.

Baader, F., Hollunder, B., Nebel, B., Profitlich, H.-J., & Franconi, E. (1994). An empirical analysis of optimization techniques for terminological representation systems. *Applied Intelligence*, 4(2):109–132.

Baader, F. & Sattler, U. (Eds.) (2000). *Proc. of the 2000 Int. Workshop on Description Logics (DL2000), Aachen, Germany, August 17-19, 2000*, volume 33 of *CEUR Workshop Proceedings* `http://ceur-ws.org/`.

Balsiger, P. & Heuerding, A. (1998). Comparison of theorem provers for modal logics - introduction and summary. In *Proc. of 2nd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX98)*, pages 25–26.

Bauknecht, K., Tjoa, A. M., & Quirchmayr, G. (Eds.) (2002). *E-Commerce and Web Technologies, Third International Conference, EC-Web 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings*, volume 2455 of *Lecture Notes in Computer Science*. Springer.

Bechhofer, S., Horrocks, I., Goble, C., & Stevens, R. (2001). OilEd: A reasonable ontology editor for the Semantic Web. In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence*, volume 2174 of *LNAI*, pages 396–408. Springer.

Bechhofer, S., Horrocks, I., Patel-Schneider, P. F., & Tessaris, S. (1999). A proposal for a description logic interface. In *Proc. of the 1999 Description Logic Workshop (DL'99)*, pages 33–36. CEUR Workshop Proceedings `http://ceur-ws.org/Vol-22/`.

Bechhofer, S., Patel-Schneider, P. F., & Turi, D. (2003a). OWL web ontology language concrete abstract syntax. Internet: `http://owl.man.ac.uk/2003/concrete/20031210`.

Bechhofer, S., Volz, R., & Lord, P. (2003b). Cooking the Semantic Web with the OWL API. In *(Fensel et al., 2003)*, pages 659–675.

Beckett, D. (2003). RDF/XML Syntax Specification (Revised). W3C Working Draft. Internet: `http://www.w3.org/TR/rdf-syntax/`.

Berners-Lee, T. (1998). Semantic web road map. Internet: `http://www.w3.org/DesignIssues/Semantic.html`.

Berners-Lee, T. (1999). *Weaving the Web*. Harper, San Francisco.

Berners-Lee, T. (2000a). CWM - closed world machine. Internet: `http://www.w3.org/2000/10/swap/doc/cwm.html`.

Berners-Lee, T. (2000b). Semantic Web talk. Invited Talk at XML 2000 Conference, Slides: `http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html`.

Berners-Lee, T., Fielding, R., Irvine, U. C., & Masinter, L. (1998). Uniform Resource Identifiers (URI): Generic Syntax. Internet Engineering Task Force (IETF), Request for Comments (RFC) 2396.

Boley, H., Tabet, S., & Wagner, G. (2001). Design rationale of RuleML: A markup language for semantic web rules. In *Proc. SWWS'01, Stanford*.

Bonner, A. J. (1992). The complexity of reusing and modifying rulebases. In *Proc. of 11th Principles of Database Systems (PODS)*, pages 316–330.

Borgida, A. (1996). On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1-2):353–367.

Borgida, A. & Brachman, R. J. (1993). Loading data into description reasoners. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 217–226. ACM Press.

Bozsak, E., Ehrig, M., Handschuh, S., Hotho, A., Maedche, A., Motik, B., Oberle, D., Schmitz, C., Staab, S., Stojanovic, L., Stojanovic, N., Studer, R., Stumme, G., Sure, Y., Tane, J., Volz, R., & Zacharias, V. (2002). KAON - Towards a Large Scale Semantic Web. In *(Bauknecht et al., 2002)*, pages 304–313. Internet: `http://link.springer.de/link/service/series/0558/bibs/2455/24550304.htm`.

Brachman, R. J. & Levesque, H. J. (1984). The tractability of subsumption in frame-based description languages. In *Proc. of the 4th National Conf. on Artificial Intelligence (AAAI-84)*, pages 34–37.

Bray, T., Hollander, D., & Layman, A. (1999). Namespaces in XML. Recommendation, W3C. Internet: `http://www.w3.org/TR/REC-xml-names/`.

Bray, T., Paoli, J., & Sperberg-McQueen, C. (2000). Extensible Markup Language (XML) 1.0. W3c recommendation, W3C. Internet: `http://www.w3.org/TR/REC-xml`.

Brickley, D. & Guha, R. V. (1999). Resource Description Framework (RDF) Schema Specification. W3C Proposed Recommendation, 03 March 1999. Internet: `http://www.w3.org/TR/1999/PR-rdf-schema-19990303/`.

Brickley, D. & Guha, R. V. (2003). RDF vocabulary description language 1.0: RDF Schema. W3C Working Draft, 10 October 2003. Internet: `http://www.w3.org/TR/2003/WD-rdf-schema-20031010/`.

Broekstra, J. & Kampman, A. (2003). Inferencing and Truth Maintenance in RDF Schema. In *In (Volz et al., 2003a)*.

Broekstra, J., Kampman, A., & van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying rdf and rdf schema. In *(Horrocks & Hendler, 2002)*, pages 54–68.

Broekstra, J., Klein, M., Decker, S., Fensel, D., van Harmelen, F., & Horrocks, I. (2001). Enabling Knowledge Representation on the Web by Extending RDF Schema. In *10th Int. WWW Conf. (WWW10)*, Hong Kong, China.

Brown, A., Fuchs, M., Robie, J., & Wadler, P. (2001). MSL - a model for XML Schema. In *WWW 10, Hong Kong, China*.

Buchheit, M., Donini, F. M., & Schaerf, A. (1993). Decidable Reasoning in Terminological Knowledge Representation Systems. *Journal of Artificial Intelligence Research*, 1:109–138.

Calvanese, D., De Giacomo, G., & Lenzerini, M. (2002). Description logics: Foundations for class-based knowledge representation. In *17th IEEE Symposium on Logic in Computer Science (LICS2002)*, pages 359–370.

Calvanese, D., Giacomo, G. D., Lenzerini, M., Nardi, D., & Rosati, R. (1998). Source integration in data warehousing. In *Proc. of 9th Int. Workshop on Database and Expert System Applications (DEXA'98)*, pages 192–197.

Carroll, J. J. & De Roo, J. (2003). OWL Web Ontology Language Test Cases. Technical report, World Wide Web Consortium (W3C).

Cattell, R. G. G., Barry, D. K., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T., & Velez, F. (Eds.) (2000). *The Object Data Standard: ODMG 3.0*. Morgen Kaufmann.

Ceri, S., Gottlob, G., & Tanca, L. (1990). *Logic Programming and Databases*. Springer, Berlin.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., & Rice, J. (1997). The generic frame protocol 2.0. Technical report, Stanford University and SRI International. Internet: `http://www.ai.sri.com/~gfp/gfp2/gfp2.html`.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., & Rice, J. (1998). OKBC: A programmatic foundation for knowledge base interoperability. In *AAAI/IAAI*, pages 600–607.

Chen, P. P. (1976). The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36.

Conen, W. & Klapsing, R. (2000). A Logical Interpretation of RDF. In *Electronic Transactions on Artificial Intelligence (ETAI), ISSN: 1401-9841*, volume 5.

Cremers, A., Griefahn, U., & Hinze, R. (1994). *Deduktive Datenbanken*. Vieweg.

Dantsin, E., Eiter, T., Gottlob, G., & Voronkov, A. (2001). Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425.

de Kleer, J. (1986). An assumption-based truth maintenance system. *Artificial Intelligence*, 28:127–162.

Dean, M. & Schreiber, G. (2003). Web Ontology Language (OWL) Reference Version 1.0. Technical report, World Wide Web Consortium (W3C).

Decker, S., Brickley, D., Saarela, J., & Angele, J. (1998). A query and inference service for RDF. In *QL98 - Query Languages Workshop*.

Decker, S., Erdmann, M., Fensel, D., & Studer, R. (1999). Ontobroker: Ontology based Access to Distributed and Semi-structured Information. In *Database Semantics: Semantic Issues in Multimedia Systems.*, pages 351–369. Kluwer Academic.

Decker, S., Fensel, D., van Harmelen, F., Horrocks, I., Melnik, S., Klein, M., & Broekstra, J. (2000). Knowledge representation on the web. In *Proc. of the 2000 Description Logic Workshop (DL 2000)*, pages 89 – 98.

Donini, F. M., Lenzerini, M., Nardi, D., & Schaerf, A. (1994). Deduction in concept languages: From subsumption to instance checking. *Journal of Logic and Computation*, 4(4):423–452.

Donini, F. M., Lenzerini, M., Nardi, D., & Schaerf, A. (1998). AL-log: Integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10:227–252.

Doyle, J. (1981). A truth maintenance system. In Webber, B. & Nilsson, N. J. (Eds.), *Readings in Artifcial Intelligence*, pages 496–516. Morgan Kaufmann, Los Altos, California.

Doyle, J. & Patil, R. S. (1991). Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48:261–297.

Eiter, T., Gottlob, G., & Mannila, H. (1997). Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418.

Elhaik, Q., Rouset, M.-C., & Ycart, B. (1998). Generating random benchmarks for description logics. In *Proc. of 1998 Description Logic Workshop (DL'98)*.

Erdmann, M. (2001). *Ontologien zur konzeptuellen Modellierung der Semantik von XML*. Books on Demand GmbH, Norderstedt, Germany.

Euzenat, J. (2000). Towards formal knowledge intelligibility at the semiotic level. In *Proc. of ECAI 2000 Workshop Applied Semiotics: Control Problems*, pages 59–61.

Fensel, D., Sycara, K., & Mylopoulos, J. (Eds.) (2003). *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, USA, October 9-12, 2002, Proceedings*, volume 2870 of *Lecture Notes in Computer Science*. Springer.

Fensel, D., van Harmelen, F., & Horrocks, I. (1999). OIL: A standard proposal for the Semantic Web. On-To-Knowledge project deliverable 0, Vrije Universiteit Amsterdam.

Fensel, D., van Harmelen, F., Horrocks, I., McGuiness, D., & Patel-Schneider, P. F. (2001). OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45.

Fikes, R. & McGuiness, D. (2001). An axiomatic semantics for RDF, RDF Schema and DAML+OIL. Technical Report KSL-01-01, KSL, Stanford University.

Forgy, C. L. (1982). RETE: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*.

Franconi, E., De Giacomo, G., MacGregor, R. M., Nutt, W., & Welty, C. A. (Eds.) (1998). *Proceedings of the 1998 International Workshop on Description Logics (DL'98), IRST, Povo - Trento, Italy, June 6-8, 1998*, volume 11 of *CEUR Workshop Proceedings*.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajararnan, A., Sagiv, Y., Ullman, J., Vassalos, V., & Widom, J. (1997). The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 2:117–132.

Genesereth, M. R. & Fikes, R. E. (1992). Knowledge Interchange Format Version 3.0 Reference Manual. Report Logic 92-1, Stanford Logic Group.

Gómez-Pérez, A. (2002). Deliverable 1.3: A survey on ontology tools. Project Deliverable 1.3, OntoWeb Thematic Network, Internet: `http://ontoweb.aifb.uni-karlsruhe.de/About/Deliverables/D13_v1-0.zip`.

Goble, C. A. (2000). What have the romans (and germans) ever done for us? or there are real applications for description logics some of which even take advantage of their reasoning services. In *(Baader & Sattler, 2000)*.

Grädel, E. (1999). On the Restraining Power of Guards. *JSL*, 64:1719–1742.

Graedel, E., Kolaitis, P. G., & Vardi, M. Y. (1997). On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69.

Grant, J. & Beckett, D. (2003). RDF Test Cases. W3C Working Draft. Internet: `http://www.w3.org/TR/rdf-testcases/`.

Grosof, B., Horrocks, I., Volz, R., & Decker, S. (2003). Description Logic Programs: Combining Logic Programs with Description Logics. In *Proc. of WWW-2003*, Budapest, Hungary.

Guha, R. V. & Hayes, P. (2003). LBase: Semantics for Languages of the Semantic Web. W3C Note, 10 October 2003. Internet: `http://www.w3.org/TR/2003/NOTE-lbase-20031010/`.

Gupta, A., Mumick, I. S., & Ross, K. A. (1995). Adapting materialized views after redefinitions. In Carey, M. J. & Schneider, D. A. (Eds.), *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 211–222. ACM Press.

Gupta, A., Mumick, I. S., & Subrahmanian, V. S. (1993). Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 157–166. ACM Press.

Haarslev, V. & Moeller, R. (2000). Expressive abox reasoning with number restrictions, role hierarchies and transitively closed roles. In *Proc. of KR 2000*, pages 273–284, San Francisco, CA, USA.

Haarslev, V. & Moeller, R. (2001). High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of IJCAI 2001*, Seattle, Washington, USA.

Haarslev, V. & Moller, R. (2001). Description of the RACER system and its applications. In *DL2001 Workshop on Description Logics, Stanford, CA*.

Handschuh, S. & Staab, S. (2002). Authoring and annotation of web pages in CREAM. In *Proc. of the 11th international conference on World Wide Web*, pages 462–473. ACM Press.

Harrison, J. & Dietrich, S. (1992). Maintenance of materialized views in a deductive database: An update propagation approach. In *Workshop on Deductive Databases held in conjunction with the Joint International Conference and Symposium on Logic Programming (JICSLP)*, pages 56–65, Washington, D.C.

Hayes, P. (2001). Rdf model theory. Technical report, W3C. Internet: `http://www.w3.org/2001/08/rdf-mt/rdf-mt`.

Hayes, P. (2003). RDF Semantics. W3C Working Draft, 10 October 2003. Internet: `http://www.w3.org/TR/rdf-mt/`.

Heflin, J., Hendler, J., & Luke, S. (1999). SHOE: A knowledge representation language for internet applications. Technical Report CS-TR-4078, Institute for Advanced Computer Studies, University of Maryland.

Heflin, J., Volz, R., & Dale, J. (2002). Requirements for a Web Ontology language. Technical report, World Wide Web Consortium (W3C).

Heinsohn, J., Kudenko, D., Nebel, B., & Profitlich, H.-J. (1994). An Empirical Analysis of Terminological Representation Systems. *Artificial Intelligence*, 68(2):367–397. 1994.

Hemaspaandra, E. (1999). The complexity of poor man's logic. In Gerbrandy, J., Marx, M., de Rijke, M., & Venema, Y. (Eds.), *Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*. Amsterdam University Press.

Hencsey, G., White, B., Chen, Y. R., Kovacs, L., & Lawrence, S. (Eds.) (2003). *Proc. of the Twelfth International World Wide Web Conference*, volume 2870. ACM.

Heuerding, A. & Schwendimann, S. (1996). A benchmark method for the propositional modal logics k, kt, s4. Technical Report IAM-96-015, University of Bern, Switzerland.

Hoelldobler, S. (1987). *Foundations of Equational Logic Programming*, volume 353 of *LNAI*. Springer.

Horrocks, I. (1997). *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester.

Horrocks, I. (1998). The FaCT System. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, pages 307–312, Oisterwijk, The Netherlands. H. de Swart.

Horrocks, I. & Hendler, J. (Eds.) (2002). *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, volume 2342 of *Lecture Notes in Computer Science*. Springer.

Horrocks, I. & Patel-Schneider, P. F. (1998). DL systems comparison. In *In (Franconi et al., 1998)*, pages 55–57.

Horrocks, I. & Patel-Schneider, P. F. (2003). Reducing OWL entailment to description logic satisfiability. In *(Fensel et al., 2003)*, pages 17–29.

Horrocks, I. & Patel-Schneider, P. F. (2003). Three theses of representation in thesemantic web. In *(Hencsey et al., 2003)*, pages 39–47.

Horrocks, I., Patel-Schneider, P. F., Boley, H., & Tabet, S. (2003a). A proposal for an OWL rules language. Draft Version of 29 October 2003 `http://www.daml.org/rules/proposal/OWL-rules-20031027/`.

Horrocks, I., Patel-Schneider, P. F., & van Harmelen, F. (2003b). From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1).

Horrocks, I. & Sattler, U. (2001). Ontology Reasoning in the $\mathcal{SHOQ}$(D) Description Logic. In Nebel, B. (Ed.), *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann.

Horrocks, I., Sattler, U., Tessaris, S., & Tobies, S. (2000). How to decide query containment under constraints using a description logic. In *Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'2000)*, Lecture Notes in Artificial Intelligence. Springer-Verlag.

Horrocks, I., Sattler, U., & Tobies, S. (1999). Practical reasoning for expressive description logics. In Ganzinger, H., McAllester, D., & Voronkov, A. (Eds.), *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, pages 161–180. Springer-Verlag.

Horrocks, I. & Tessaris, S. (2002). Querying the web: A formal approach. In *(Horrocks & Hendler, 2002)*, pages 177–191.

Horrocks, I., van Harmelen, F., Patel-Schneider, P., Berners-Lee, T., Brickley, D., Connolly, D., Dean, M., Decker, S., Fensel, D., Hayes, P., Heflin, J., Hendler, J., Lassila, O., McGuinness, D., & Stein, L. A. (2001). Daml+oil (march 2001). Internet: http://www.daml.org/2001/03/daml+oil-index.

Immerman, N. (1986). Relational queries computable in polynomial time. *Information and Control*, 68(1-3):86–104.

Kifer, M., Lausen, G., & Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843.

Klyne, G. & Carroll, J. (2003). Resource Description Framework (RDF): Concepts and Abstract Data Model. W3C Working Draft. Internet: `http://www.w3.org/TR/rdf-concepts/`.

Kohlmayer, R. (2000). Vorsicht! bissiger mund! *Die Schnake (ISSN 0723-7227)*, 15+16.

Kowalski, R. A. & Kuehner, D. (1971). Linear resolution with selection function. *Artificial Intelligence*, 2(3/4):227–260.

Kuchenhoff, V. (1991). On the efficient computation of the difference betwen consecutive database states. In Delobel, C., Kifer, M., & Masunaga, Y. (Eds.), *Proc. of 2nd Int. Conf. on Deductive and Object-Oriented Databases*, volume 566 of *Lecture Notes in Computer Science (LNCS)*, pages 478–502, Munich, Germany. Springer.

Lassila, O. & Swick, R. (1999). Resource Description Framework (RDF) Model and Syntax Specification. W3C Working Draft. Internet: `http://www.w3.org/TR/REC-rdf-syntax/`.

Leone, N., Pfeifer, G., Faber, W., Calimeri, F., Dell'Armi, T., Eiter, T., Gottlob, G., Ianni, G., Ielpa, G., Koch, C., Perri, S., & Polleres, A. (2002). The dlv system. In *Proc. of the 8th European Conf. on Artificial Intelligence (JELIA)*, pages 537–540. Springer.

Levinson, R. A. (1992). Pattern associativity and the retrieval of semantic networks. *Journal of Computers + Mathematics with Applications*, 23(6-9):573–600.

Levy, A. Y. & Rousset, M.-C. (1996). CARIN: A Representation Language Combining Horn Rules and Description Logics. In Wahlster, W. (Ed.), *Proc. of 12th European Conference on Artificial Intelligence*, pages 323–327, Budapest, Hungary. John Wiley and Sons, Chichester.

Lipkis, T. (1982). A KL-ONE classifier. In Schmolze, J. G. & Brachman, R. J. (Eds.), *Proc. of 1981 KL-ONE Workshop*, pages 128–145, Cambridge, Mass. The proceedings have been published as BBN Report No. 4842 and Fairchild Technical Report No. 618.

Liu, L., Pu, C., & Tang, W. (1999). Continual queries for internet scale event-driven information delivery. *IEEE TKDE*, 11(4).

Lloyd, J. W. (1987). *Foundations of Logic Programming (second, extended edition)*. Springer series in symbolic computation. Springer-Verlag, New York.

MacGregor, R. (1988). A deductive pattern matcher. In *Proc. of the 7th Nat. Conf. of the Am. Assoc. for Artificial Intelligence (AAAI)*, pages 403–408.

MacGregor, R. (1991). Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92.

Maedche, A., Motik, B., Silva, N., & Volz, R. (2002a). MAFRA - A Mapping Framework for Distributed Ontologies. In *Proc. of 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW)*, Siquenca, Spain.

Maedche, A., Motik, B., Stojanovic, L., Studer, R., & Volz, R. (2002b). Managing multiple ontologies and ontology evolution in ontologging. In *Proc. of IIP-2002*, Montreal, Canada.

Maedche, A., Motik, B., Stojanovic, L., Studer, R., & Volz, R. (2003). An infrastructure for searching, reusing and evolving distributed ontologies. In *Proc. of WWW-2003*, Budapest, Hungary.

Maier, D. & Warren, D. S. (1988). *Computing with Logic: Logic Programming with Prolog*. Benjamin/Cummings Publishing Co., Menlo Park, CA, USA.

Manola, F. & Miller, E. (2003). RDF Primer. W3C Working Draft. Internet: `http://www.w3.org/TR/rdf-primer/`.

McBride, B. (2001). Jena: Implementing the rdf model and syntax specification. In *Proceedings of the Second International Workshop on the Semantic Web - SemWeb'2001, Hongkong, China, May 1, 2001*.

McGuiness, D. & da Silva, P. P. (2003). Infrastructure for web explanations. In *(Fensel et al., 2003)*, pages 113–129.

McGuinness, D. L. & van Harmelen, F. (2002). Feature Synopsis for OWL Lite and OWL. Technical report, World Wide Web Consortium (W3C).

McGuinness, D. L. & van Harmelen, F. (2003). OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium (W3C). Internet: `http://www.w3.org/TR/owl-features/`.

Motik, B., Maedche, A., & Volz, R. (2002a). A conceptual modeling approach for semantics-driven enterprise applications. In *Proc. 1st Int'l Conf. on Ontologies, Databases and Application of Semantics (ODBASE-2002)*.

Motik, B., Maedche, A., & Volz, R. (2002b). A conceptual modeling approach from semantics-driven enterprise applications. In *Proc. of ODBASE 2002*, Newport, California.

Motik, B., Oberle, D., Staab, S., Studer, R., & Volz, R. (2002c). KAON SERVER Architecture. Technical Report 421, Institute AIFB, University of Karlsruhe, D-76128 Karlsruhe, Germany. Internet: `http://www.aifb.uni-karlsruhe.de/WBS/dob/pubs/D5.pdf`.

Motik, B., Volz, R., & Mädche, A. (2003). Optimizing Query Answering in Description Logics using Disjunctive Deductive Databases. In *KRDB Workshop Colocated with KI 2003*, Hamburg, Germany. Internet: `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-79/motik.pdf`.

Nayak, P. P. & Williams, B. C. (1998). Fast context switching in real-time propositional reasoning. In Senator, T. & Buchanan, B. (Eds.), *Proceedings of the Fourteenth National Conference on Artificial Intelligence and the Ninth Innovative Applications of Artificial Intelligence Conference*, pages 50–56, Menlo Park, California. AAAI Press.

Nebel, B. (1990). Terminological Reasoning is Inherently Intractable. *Artificial Intelligence*, 43:235–249.

Nebel, B. (1991). Terminological Cycles: Semantics and Computational Properties. In Sowa, J. F. (Ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 331–361. Morgan Kaufmann Publishers, San Mateo (CA), USA.

Nejdl, W., Dhraief, H., & Wolpers, M. (2001). O-Telos-RDF: A resource description format with enhanced metamodeling functionalities based on O-Telos. In *Workshop on Knowledge Markup and Semantic Annotation at the First Int. Conf. on Knowledge Capture (K-CAP 2001), Canada*.

Network Inference Inc. (2003). Cerebra server datasheet. Internet: `http://www.networkinference.com/Assets/Products/Cerebra_Server_Datasheet.pdf`. Last Checked on 13.12.03.

Noy, N., Fergerson, R., & Musen, M. (2000). The knowledge model of Protégé-2000: Combining interoperability and flexibility. In *Proc. of 12th Int. Conf. on Knowledge Engineering and Knowledge Management: Methods, Models, and Tools (EKAW 2000)*, pages 17–32.

Noy, N. F. & Musen, M. A. (2000). PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proc. of AAAI-2000*, pages 450–455.

Object Management Group (2003). Omg unified modeling language specification. Internet: `http://www.omg.org/technology/documents/formal/uml.htm`. Version 1.5, 1. March 2003.

Omelayenko, B. (2002). Integrating Vocabularies: Discovering and Representing Vocabulary Maps. In *Proceedings of the First International Semantic Web Conference (ISWC-2002), Sardinia, Italy*.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference*, pages 161–172, Brisbane, Australia.

Pan, J. & Horrocks, I. (2001). Metamodeling architecture of web ontology languages. In *Proc. of the First Semantic Web Working Symposium (SWWS'01)*, pages 131–149.

Pan, J. Z. & Horrocks, I. (2002). Reasoning in the shoq(dn) description logic. In *Proceedings of the 2002 International Workshop on Description Logics (DL2002)*, volume 52. CEUR Workshop Proceedings.

Patel-Schneider, P. F. & Fensel, D. (2002). Layering the semantic web: Problems and directions. In *(Horrocks & Hendler, 2002)*, pages 16–29.

Patel-Schneider, P. F., Hayes, P., & Horrocks, I. (2003). Web Ontology Language Semantics and Abstract Syntax. W3C Candidate Recommendation, Version 18 August 2003. Internet: `http://www.w3.org/TR/owl-semantics/`.

Patel-Schneider, P. F., McGuiness, D. L., Brachman, R. J., Resnick, L. A., & Borgida, A. (1991). The CLASSIC knowledge representation system: Guiding principles and implementation rationale. *SIGART Bulletin*, 2(3):108–113.

Plaisted, D. A. & Greenbaum, S. (1986). A structure-preserving clause form transformation. *Journal of Symbolic Logic and Computation*, 2(3):293–304.

Ramakrishnan, R. (1991). Magic templates: A spellbinding approach to logic programs. *Journal of Logic Programming*, 11:189–216.

Ramakrishnan, R., Srivastava, D., Sudarshan, S., & Seshadri, P. (1994). The CORAL Deductive System. *VLDB Journal: Very Large Data Bases*, 3(2):161–210.

Rector, A. L., Zanstra, P. E., Solomon, W. D., Rogers, J. E., Baud, R., & Wagner, J. (1999). Reconciling users' needs and formal requirements: Issues in developing a re-usable ontology for medicine. *IEEE Transactions on Information Technology in BioMedicine*, 2(4):229–242.

Richardson, M., Agrawal, R., & Domingos, P. (2003). Trust Management for the Semantic Web. In *(Fensel et al., 2003)*, pages 351–368.

Robinson, J. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41.

Roo, J. D. (2002). Euler proof mechanism. Internet: `http://www.agfa.com/w3c/euler/`.

Rousset, M. (2002). Standardization of a web ontology language. *IEEE Intelligent Systems, March/April 2002*.

Sagonas, K., Swift, T., & Warren, D. S. (1994). XSB as an efficient deductive database engine. In Snodgrass, R. T. & Winslett, M. (Eds.), *Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94)*, pages 442–453.

Schaerf, A. (1994). Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176.

Schild, K. (1991). A correspondence theory for terminological logics: preliminary report. In *Proceedings of IJCAI-91, 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sidney, Australia.

Schmidt-Schauß, M. (1989). Subsumption in KL-ONE is undecidable. In *Proc. of KR 89*, pages 421–431. Morgan Kaufmann.

Schmidt-Schauß, M. & Smolka, G. (1991). Attributive Concept Descriptions with Complements. *AIJ*, 48(1):1–26.

Schöning, U. (1995). *Logik für Informatiker*. Spektrum Akademischer Verlag.

Simeon, J. & Wadler, P. (2003). The essence of xml. In *Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 1–13. ACM Press.

Sintek, M. & Decker, S. (2001). TRIPLE - an RDF query, inference and transformation language. In *Deductive Databases and Knowledge Management (DDLP)*.

Smith, M. K., McGuinness, D., Volz, R., & Welty, C. (2003). Web Ontology Language (OWL) Guide. Technical report, World Wide Web Consortium (W3C). Internet: [http://www.w3.org/TR/2003/CR-owl-guide-20030818/.

Smolka, G. (1988). A feature logic with subsorts. Technical Report 33, IWBS, IBM Deutschland, Postfach 800880, D-7000 Stuttgart 80, Germany.

Spyns, P., Oberle, D., Volz, R., Zheng, J., Jarrar, M., Sure, Y., Studer, R., & Meersman, R. (2002). OntoWeb - A Semantic Web community portal. In *Proc. ofProc. Fourth International Conference on Practical Aspects of Knowledge Management (PAKM)*, pages 189–200, Vienna, Austria.

Staudt, M. & Jarke, M. (1995). Incremental maintenance of externally materialized views. Technical Report AIB-95-13, RWTH Aachen.

Staudt, M. & Jarke, M. (1996). Incremental maintenance of externally materialized views. In Vijayaraman, T. M., Buchmann, A. P., Mohan, C., & Sarda, N. L. (Eds.), *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 75–86. Morgan Kaufmann.

Stojanovic, L., Stojanovic, N., & Volz, R. (2002). Migrating data-intensive web sites into the semantic web. In *Proc. of ACM Symposium on Applied Computing (SAC)*, Madrid, Spain.

Studer, R., Benjamins, R., & Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data + Knowledge Engineering*, 25(1-2):161–197.

Studer, R., Sure, Y., & Volz, R. (2002). Managing user focused access to distributed knowledge. *Journal of Universal Computer Science (J.UCS)*, 8(6):662–672.

Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., & Wenke, D. (2002). Ontoedit: Collaborative ontology development for the semantic web. In *Proceedings of the 1st International Semantic Web Conference (ISWC2002), June 9-12th, 2002, Sardinia, Italia*. Springer.

Swift, T. & Warren, D. S. (1994). Analysis of SLG-WAM Evaluation of Definite Programs. In *Symposium on Logic Programming*, pages 219–235.

Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309.

Tarski, A. (1956). *Logic, Semantics, Metamathematics*. Oxford University Press.

Tempich, C. & Volz, R. (2003). Towards a benchmark for Semantic Web reasoners. In *Proc. of the 2nd International Workshop on Evaluation of Ontology-based Tools , collocated with the 2nd International Semantic Web Conference ISWC 2003*, Sundial Resort, Sanibel Island, Florida, USA.

The Unicode Consortium (2003). *The Unicode Standard, Version 4.0.0*. Addison-Wesley.

Thompson, H., Beech, D., Maloney, M., & Mendelsohn, N. (2001). XML Schema part 1: Structures. Recommendation, W3C. http://www.w3.org/TR/xmlschema-1/.

Tolle, K. (2000). Validating RDF Parser: A Tool for Parsing and Validating RDF Metadata and Schemas. Master's thesis, University of Hannover.

Ullman, J. D. (1988). *Principles of Database and Knowledge-base Systems*, volume 1. Computer Science Press.

van Benthem, J. (1984). Correspondence theory. In Gabbay, D. M. & Guenther, F. (Eds.), *Handbook on Philosophical Logic*, volume 2, pages 167–247. D. Reidel Publishing Company.

van Emden, M. H. & Kowalski, R. A. (1976). The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742.

van Gelder, A. (1989). The alternative fixpoint of logic programs with negation. In *Proc. of Principles of Database Systems (PODS'89)*, pages 1–10, Philadelphia, Pennylvania, USA.

van Gelder, A., Ross, K. A., & Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650.

Vardi, M. Y. (1982). The complexity of relational query languages (extended abstract). In *Proc.of 14th ACM Symp. on Theory of Computing (STOC '82)*, pages 341–351, San Francisco, California.

Volz, R. (2003). External ontologies in the semantic web. In *Proc. of the 20th British National Conference on Databases (BNCOD)*, Coventry, UK.

Volz, R., Decker, S., & Cruz, I. (Eds.) (2003a). *Proc. of first Int. Workshop on Practical and Scalable Semantic Systems*, volume 89, Sanibel Island, Florida, USA. CEUR Workshop Proceedings. Internet: `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-89/`.

Volz, R., Decker, S., & Oberle, D. (2002a). Bubo - Implementing OWL in rule-based systems. Internet: WebOnt Working Group Mail Archive. Submission to WWW2003, merged with (Grosof et al., 2003).

Volz, R., Handschuh, S., Staab, S., Stojanovic, L., & Stojanovic, N. (2004). Unveiling the hidden bride: Deep Annotation for Mapping and Migrating Legacy Data to the Semantic Web. *Web Semantics*. to appear.

Volz, R., Motik, B., Horrocks, I., & Grosof, B. (2003b). Description logics programs: An evaluation and extended translation. Report on a first performance evaluation, available from `http://kaon.semanticweb.org/`.

Volz, R., Oberle, D., Staab, S., & Motik, B. (2003c). KAON SERVER - a semantic web management system. In *Proc. of WWW-2003*, Budapest, Hungary.

Volz, R., Oberle, D., Staab, S., & Studer, R. (2002b). OntoLift prototype. Technical Report D11, WonderWeb project deliverable.

Volz, R., Oberle, D., & Studer, R. (2002c). Towards views in the semantic web. In *2nd International Workshop on Databases, Documents and Information Fusion (DBFUSION02)*, Karlsruhe, Germany.

Volz, R., Oberle, D., & Studer, R. (2003d). Implementing views for light-weight web ontologies. In *Proc. of Int. Database Engineering and Application Symposium (IDEAS)*, Hong Kong, China.

Volz, R., Oberle, D., & Studer, R. (2003e). Views for light-weight web ontologies. In *Proc. of ACM Symposium of Applied Computing (SAC)*, Melbourne, Florida, USA.

Volz, R., Staab, S., & Motik, B. (2003f). Incremental maintenance of dynamic datalog programs. In *(Volz et al., 2003a)*.

Volz, R., Staab, S., & Motik, B. (2003g). Incremental Maintenance of Materialized Ontologies. In Meersman, R., Tari, Z., Schmidt, D. C., Kraemer, B., van Steen, M., Vinoski, S., King, R., Orlowska, M., Studer, R., Bertino, E., & McLeod, D. (Eds.), *Proc. of CoopIS/DOA/ODBASE 2003*, volume 2888 of *LNCS*, pages 707–724, Sicily.

W3C (2001). W3C Semantic web activity statement. Internet: `http://www.w3.org/2001/sw/Activity`.

W3C (2003). Web ontology (webont) working group charter. Internet: `http://www.w3.org/2002/11/swv2/charters/WebOntologyCharter`.

Weithöner, T., Liebig, T., & Specht, G. (2003). Storing and Querying Ontologies in Logic Databases. In *Proceedings of The first International Workshop on Semantic Web and Databases (SWDB'03)*, pages 329 – 348, Berlin, Germay.

Zou, Y. (2001). DAML XSB interpretation. Version 0.3.