National Technical University of Athens

School of Electrical
and Computer Engineering

Division of Computer Science

Optimizing Query Answering over Expressive Ontological Knowledge

DOCTOR OF PHILOSOPHY

of

ILIANNA S. KOLLIA

Electrical and Computer Engineer N.T.U.A. (2009)

Athens, April 2014

National Technical University of Athens
School of Electrical & Computer Engineering
Division of Computer Science

Optimizing Query Answering over Expressive Ontological Knowledge

DOCTOR OF PHILOSOPHY

of

ILIANNA S. KOLLIA

Electrical and Computer Engineer N.T.U.A. (2009)

Advisory Commitee: Andreas-Georgios Stafylopatis
Birte Glimm
Giorgos Stamou

... A.-G. Stafylopatis Professor N.T.U.A.	... B. Glimm Assistant Professor University of Ulm	... G. Stamou Assistant Professor N.T.U.A.
... P. Tsanakas Professor N.T.U.A.	... M. Koubarakis Professor U.O.A.	... K. Kontogiannis Associate Professor N.T.U.A.
	... I. Horrocks Professor University of Oxford	

Athens, April 2014

. . .

ILIANNA S. KOLLIA
Doctor Electrical and Computer Engineer N.T.U.A.

The approval of the Ph.D. Thesis from the School of Electrical and Computer Engineering of N.T.U.A. does not indicate acceptance of the opinions of the author (L.5343/1932, Article 202).

Contents

List of Figures

List of Tables

ABSTRACT

Query answering over ontologies, i.e., the computation of answers to user queries based not only on explicitly stated information but also on implicit knowledge is an important task in the context of the Semantic Web. In this direction, the SPARQL query language has recently been extended by the World Wide Web Consortium (W3C) with so-called *entailment regimes*. An entailment regime defines how queries are evaluated under more expressive semantics than SPARQL's standard simple entailment, which is based on subgraph matching.

In this thesis we describe a sound and complete algorithm for the OWL Direct Semantics entailment regime of SPARQL (SPARQL-OWL). The proposed SPARQL-OWL queries are very expressive since variables can occur within complex concepts and can also bind to concept or role names apart from individuals. Initially, we present a cost-based query planning strategy for SPARQL queries issued over an OWL ontology. The costs of the model are based on information about the instances of concepts and roles that are extracted from a model abstraction built by an OWL reasoner. A static and a dynamic algorithm are presented which use these costs to find optimal or near optimal execution orders for the templates of a query. For the dynamic case, we improve the performance by exploiting an individual clustering approach that allows for computing the cost functions based on one individual sample per cluster. Afterwards, we propose optimizations that target particularly the complex queries that are allowed in SPARQL-OWL. These optimizations exploit query rewriting techniques and the concept and role hierarchies to efficiently answer such queries.

The proposed algorithm and optimizations have been implemented in a system called OWL-BGP. Our experimental study, using this system, shows that the static ordering usually outperforms the dynamic one when accurate statistics are available. This changes, however, when the statistics are less accurate, e.g., due to non-deterministic reasoning decisions. For complex SPARQL-OWL queries we observe an improvement of up to three orders of magnitude due to the proposed optimizations. Finally, we show that the implemented system works well in a real world application about cultural heritage data.

Keywords: SPARQL query answering, SPARQL-OWL, OWL Direct Semantics entailment regime, query planning, query optimization

Acknowledgements

I would like to thank my supervisor Professor Andreas-Georgios Stafylopatis for his support and valuable help whenever needed during my Ph.D. studies. Moreover, I would like to thank my second supervisor Birte Glimm, Assistant Professor at the University of Ulm in Germany, for her continuous help and support both when working for my Ph.D. under her guidance in Ulm and through our weekly Skype meetings. I would also like to thank her, as well as Dr. Yevgeny Kazakov, for being so eager to help me whenever some problem arose during my whole stay in Ulm. I would also like to thank the third member of my Ph.D. Advisory Committee Assistant Professor Giorgos Stamou for the continuous interaction we had during all this period. All three created a friendly, high-level working environment from the beginning to the end of my Ph.D. and I truly thank them for this.

I would also like to give my sincere thanks to Professor Ian Horrocks, who was the supervisor of my M.Sc. Thesis at the University of Oxford, and Professor Manolis Koubarakis of the University of Athens for accepting being external members of my Ph.D. Committee. Special thanks to Professor Panayiotis Tsanakas for his assistance during all my Ph.D. studies and to Associate Professor Kostas Kontogiannis for participating in my Ph.D. Examination Committee.

I would also like to thank all my friends and colleagues, who provided me with good company and a nice working environment that were of great assistance to my working in a calm and dedicated manner. Last but not least, I thank my family -my parents, Stefanos and Loula and my brother Dimitris- for the continuous help and support they provided during all my studies.

Abbreviations

ABox	:	Assertional Box
b.c.	:	before Christ
BGP	:	Basic Graph Pattern
CIDOC CRM	:	CIDOC Conceptual Reference Model
CPU	:	Central Processing Unit
DC	:	Dublin Core
DL	:	Description Logics
DL-KB	:	Description Logic Knowledge Base
DLP	:	Description Logic Program
EAD	:	Encoded Archival Description
EDM	:	Europeana Data Model
e.g.	:	for example
ESE	:	Europeana Semantic Elements
etc.	:	et cetera
FIFO	:	First In First Out
FOL	:	First Order Logic
FSS	:	Functional Style Syntax
GCI	:	General Concept Inclusion
IH	:	Induction Hypothesis
I/O	:	Input/Output
IRI	:	International Resource Identifier
KB	:	Knowledge base
LIDO	:	Lightweight Information Describing Objects
LUBM	:	Lehigh University Benchmark
METS	:	Metadata Encoding and Transmission Standard
MPEG	:	Motion Pictures Experts Group
OWL	:	Web Ontology Language
RDF	:	Resource Description Framework
RDFS	:	RDF Schema
SKOS	:	Simple Knowledge Organization System
SURF	:	Speeded Up Robust Features
SVM	:	Support Vector Machine
TBox	:	Terminological Box
UOBM	:	University Ontology Benchmark
w.r.t.	:	with respect to
WWW	:	World Wide Web
W3C	:	World Wide Web Consortium
XML	:	Extensible Markup Language

Chapter 1

Introduction

The last couple of years the area of *knowledge representation and reasoning* [15]—a subfield of *Artificial Intelligence*—has gained much attention. This area is centered around the way knowledge should be organized and processed so that it can be used by reasoning procedures to extract useful conclusions. A knowledge representation system consists of a knowledge base (KB), where constraints and facts about the modeled world are stored. There are several knowledge representation formalisms proposed in the literature; in the current thesis we focus on *Description Logics* (DLs), which are a decidable fragment of First Order Logic (FOL) and the logical foundation of the Web Ontology Language (OWL); a language widely used in the Semantic Web [13, 118]. The Semantic Web is an extension of the World Wide Web - WWW, in which information is organized in such a way that it is processable and understandable by computer programs, which can carry out complex tasks in a (semi-)automatic way. Description Logic systems have been widely used in several practical applications including medical informatics [111, 142, 128, 122, 38], life sciences [141, 120, 140, 115, 130, 1], cultural heritage [81, 80] and image processing and multimedia [9, 97, 109, 79].

Description Logic and generally knowledge representation systems give users the ability to ask (complex) queries over the stored knowledge and, through appropriate algorithms, to retrieve answers to them based not only on explicitly stated facts, but also on implicit or inferred knowledge. Several methods have been developed for query answering over Description Logic systems, most of which focus on conjunctive queries well known from the area of databases [2]. Since such systems either work with simple description logics or are not practical enough for logics of higher expressivity, the aim of this thesis is to propose a novel Description Logic query language that allows higher order features and to develop a query answering algorithm together with optimizations for answering such queries over expressive Description Logic knowledge bases.

This chapter gives an informal introduction to the formalisms used in the thesis. Section 1.1 introduces Description Logics and the used query language, which we call *SPARQL-OWL*. The intuition for this query language comes from SPARQL [110], a query language widely used in the area of RDF triple stores; in fact, every SPARQL-OWL query can be mapped to a SPARQL query and vice versa. In Section 1.2 the connection between the two languages is made clear by explaining how a query issued in SPARQL can be transformed to a query in the proposed SPARQL-OWL language. Note that the tool that has been developed in the context of the thesis

takes SPARQL queries as input and using this translation it produces SPARQL-OWL queries which are afterwards evaluated using the query answering algorithm and optimizations proposed in the thesis.

1.1 Description Logics

Description Logics [7] are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way. Like other logical formalisms they define the *syntax* and *semantics* of modeled information. Syntax refers to the definition of the symbols used by the formalism, as well as the rules that combine these symbols to more complex constructs while semantics refers to the definition of the interpretations given to the syntactic symbols as well as the interpretations of their combinations. Description logics are a decidable fragment of First Order Logic [28] and are widely used in several domains.

A Description Logic *knowledge base* (DL-KB), usually called *ontology*, typically consists of two parts, the *terminological* and the *assertional* part. In the terminological part, called *TBox* a vocabulary of the used terms is provided, together with axioms describing the meaning of these terms. The assertional part, called *ABox*, is used to state assertions about concrete data. The main building blocks of DLs are *concept names*, *(abstract) role names* and *individual names* or *individuals*. For example, in a university domain we can have individual names like *mary*, *maths*, concept names like Person, Student, UndergraduateStudent, Professor, Course, GraduateCourse and role names like takesCourse, teachesCourse. (Complex) concepts in DLs are built using concept, role and individual names that are connected through constructors. The semantics of DLs is given by means of interpretations. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set of objects $\Delta^{\mathcal{I}}$ called the *interpretation domain*, and an *interpretation function* $\cdot^{\mathcal{I}}$. The interpretation function maps concept names to sets of objects in $\Delta^{\mathcal{I}}$, role names to binary relations between objects in $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and individual names to objects in $\Delta^{\mathcal{I}}$. Using the semantics of concept, role and individual names the semantics of more complex concepts can be determined [7]. For example, assume that we want to define the concept of a student that takes a graduate course. This can be described by the following concept:

$$\mathsf{Student} \sqcap \exists \mathsf{takesCourse}.\mathsf{GraduateCourse} \tag{1.1}$$

Informally, in an interpretation $\mathcal{I}$, the concept name Student (GraduateCourse) is mapped to the set containing all objects that are students (graduate courses) and the role name takesCourse is mapped to the set of pairs of objects; the first element of each pair is an object that takes a course and the second element is the course that it takes. The concept uses *conjunction* ($\sqcap$), which is interpreted as set intersection and an *existential restriction*. An individual, say *mary*, belongs to the existential restriction ($\exists$takesCourse.GraduateCourse) if there is an individual that is a GraduateCourse and is related to *mary* via the takesCourse role. Depending on the used constructors there are DLs of different expressivity.

Axioms contain, on the one hand, general information about the modeled domain, i.e., the relationships between concept names and role names and, on the other hand, information about concrete instance data, i.e., how individuals are related to

concept names and to other individuals through role names. For example, assume that we want to state that students are either undergraduate students or they take at least 3 graduate courses. This statement is represented by the following concept inclusion axiom:

$$\mathsf{Student} \sqsubseteq \mathsf{UndergraduateStudent} \sqcup \geqslant 3\ \mathsf{takesCourse.GraduateCourse} \tag{1.2}$$

The example above contains a *disjunction* constructor ($\sqcup$) and a *qualified number restriction* ($\geqslant 3$ takesCourse.GraduateCourse). From a semantics point of view, we say that Axiom (1.2) is *satisfied* in an interpretation $\mathcal{I}$, if it holds in $\mathcal{I}$ that the set of students is a subset of the set consisting of the individuals that belong either to i) the set of undergraduate students or to ii) the set of individuals that are connected via the role takesCourse to at least three individuals from the set of graduate courses or to iii) both sets. Examples of assertional axioms are shown below:

$$\mathsf{UndergraduateStudent}(mary) \tag{1.3}$$
$$\mathsf{takesCourse}(mary, maths) \tag{1.4}$$

Axiom (1.3) is a concept assertion and states that *mary* is an UndergraduateStudent. Axiom (1.4) is a role assertion and states that *mary* is connected to *maths* via the takesCourse role. Semantically, Axiom (1.3) is satisfied in an interpretation $\mathcal{I}$ if *mary* (more correctly the object in $\Delta^{\mathcal{I}}$ to which *mary* is mapped) belongs to the set of undergraduate students in $\mathcal{I}$. Similarly, one can define when an interpretation $\mathcal{I}$ satisfies Axiom (1.4). An interpretation that satisfies all axioms of a knowledge base is called a *model*.

The computation of inferences by a knowledge base system is called *reasoning*. The standard reasoning services provided by description logic systems are the following:

- Knowledge base consistency checking: to check whether a knowledge base contains a contradiction. A contradiction arises when a fact and the negation of the fact can both be derived by the axioms of the knowledge base.

- Concept satisfiability checking: to check, given a knowledge base and a concept, whether there is a model of the knowledge base in which the given concept is interpreted to a non-empty set of objects.

- Concept subsumption checking: to check, given a knowledge base and two concepts, whether for every model $\mathcal{I}$ of the knowledge base, the set of objects to which $\mathcal{I}$ interprets the first given concept is a subset of the set of objects to which $\mathcal{I}$ interprets the second given concept. Subsumption checking is used to build a taxonomy of concept names, called *concept hierarchy*, through a process called *classification*.

- Instance checking: to check, given a knowledge base, a concept (role) and an individual (pair of individuals) whether for every model $\mathcal{I}$ of the knowledge base the object to which the individual (the pair of objects to which the pair of individuals) is interpreted by $\mathcal{I}$ is an element of the set the concept (role) is interpreted by $\mathcal{I}$.

It has been shown that all these reasoning services can be transformed to KB consistency checking [7]. As it has been stated before, the reasoning problems in DLs are decidable. The complexity of these problems, however, depends on the DL under consideration. The more expressive a DL is, the more complex the inference problems are in this DL, i.e., there is a trade-off between expressivity of a DL and complexity of the reasoning tasks in this DL. The presence of disjunctive axioms like Axiom (1.2) in a knowledge base often makes reasoning problems "harder" in this logic. For example, consistency checking (and hence all standard reasoning problems) in the description logic $\mathcal{SROIQ}$, which is the description logic underpinning OWL 2 DL (and the language of interest in this thesis) is 2-NExpTime complete [72].

There are many reasoning algorithms developed in the literature for DLs of different expressivity [61, 59, 60, 58, 95, 96]. The most commonly used *(hyper)tableau* reasoning algorithms test the consistency of a knowledge base by constructing a model or else by proving that there is no model; i.e., there is a contradiction. Because of the high worst case complexity of reasoning over expressive DLs, currently used algorithms employ several optimizations to achieve practicality. In practical systems, consistency checking is usually performed by tools called *reasoners*. There are several reasoners for expressive DLs, like Racer Pro [47], Fact++ [136], Pellet [126], or HermiT [119].

Another important service that knowledge representation systems provide is that of answering queries. When computing answers to queries, such systems do not only return the explicitly given knowledge, but they also take into account the implicit knowledge coming from the inference procedure. In highly expressive DLs query answering is usually based on consistency checking and is thus a task of high computational complexity [34, 32, 30, 101]. Work has only recently begun on optimizing the query answering services of DL systems. Note that in knowledge bases, in contrast to databases, the open world assumption is adopted, according to which it is assumed that the knowledge we have over the domain of interest is incomplete; absence of information does not mean negative information. More details about this assumption and consequences it entails are given in Section 3.2.

Since the focus in this thesis is the optimization of query answering over expressive Description Logics such as $\mathcal{SROIQ}$, the description of SPARQL-OWL, the used query language, is given next. In the proposed language, standard conjunctive instance queries, i.e., conjunctive queries which allow only distinguished variables (conjunctive queries without existential variables) can be expressed as well as complex queries, which allow for higher order features; variables in queries can appear not only in place of individuals, like in standard conjunctive queries, but also in place of concepts and roles. The answers to such queries consist of mappings of variables to concept, role and individual names from the queried knowledge base such that, after replacing the query variables with the corresponding concept, role or individual names from the variable mappings, all query conjuncts are true in every model of the knowledge base.

For example, one can ask a standard conjunctive query for the chairs of a university and the graduate courses that they teach:

$$q = \{\mathsf{Chair}(?x), \mathsf{teaches}(?x, ?y), \mathsf{GraduateCourse}(?y)\},$$

where Chair, GraduateCourse are concept names, teaches is a role name and $?x$,$?y$

are variables. Note that it is not necessary to enumerate the variables in the query head. If we assume that the queried knowledge base contains the following TBox and ABox:
$\mathcal{T} = \{$

$$\mathsf{Person} \sqcap \exists \mathsf{isHeadOf}.\mathsf{Department} \sqsubseteq \mathsf{Chair} \tag{1.5}$$
$$\mathsf{Course} \sqcap \exists \mathsf{takesCourse}^{-}.\mathsf{GraduateStudent} \sqsubseteq \mathsf{GraduateCourse} \tag{1.6}$$

$\}$

$\mathcal{A} = \{$

$$\mathsf{GraduateStudent}(mary) \tag{1.7}$$
$$\mathsf{takesCourse}(mary, maths) \tag{1.8}$$
$$\mathsf{Person}(tom) \tag{1.9}$$
$$\mathsf{Course}(maths) \tag{1.10}$$
$$\mathsf{teaches}(tom, maths) \tag{1.11}$$
$$\mathsf{isHeadOf}(tom, csdep) \tag{1.12}$$
$$\mathsf{Department}(csdep) \tag{1.13}$$

$\}$
then the answer to the above query q is the mapping $?x \mapsto tom$, $?y \mapsto maths$, since the assertion $\mathsf{Chair}(tom)$ is a consequence of Axioms (1.5), (1.9), (1.12) and (1.13), $\mathsf{teaches}(tom, maths)$ is explicitly given in the ABox and $\mathsf{GraduateCourse}(maths)$ is a consequence of Axioms (1.6), (1.7), (1.8) and (1.10) (note that $\mathsf{takesCourse}^{-}$ is the inverse of the role takesCourse, i.e., if $\mathsf{takesCourse}^{-}(a, b)$ holds then it also holds $\mathsf{takesCourse}(b, a)$).

Another type of valid query in the language, that asks for the individuals that take more than four graduate courses, is the following:

$$q = \{\geq 4\ \mathsf{takesCourse}.\mathsf{GraduateCourse}(?x)\}$$

The following is a query in which variable $?x$ appears in place of a concept:

$$q = \{\mathsf{Professor} \sqsubseteq \exists \mathsf{isHeadOf}.?x\}$$

This query asks for the concepts whose instances are individuals to which every instance of the concept Professor is related via the role isHeadOf.

In the rest of the thesis we usually call such queries with variables in nested concept and role positions *complex queries* or *queries with complex axiom templates*. Moreover, we use the term ontology instead of knowledge base.

1.2 Semantic Web Standards

In this section we briefly discuss some of the Semantic Web standards for knowledge representation and query answering [53], since, in the current thesis, we use a Description Logic query language that directly corresponds to an extension of the SPARQL query language [110].

Several standards for ontologies and queries have been developed in the context of the Semantic Web, including the Resource Description Framework (RDF), RDF Schema (RDFS), the Web Ontology language (OWL), its extension OWL 2 and SPARQL. RDF [90] is used for the representation of information about web resources in the form of triples consisting of a *subject*, a *predicate* and an *object*. RDF terms can be either *International Resource Identifiers* (IRIs), or *blank nodes* or *literals*. Blank nodes in RDF denote the existence of a resource without explicitly stating its name, while literals are strings of characters. The subject of triples can be either an IRI or a blank node, while the object can be any RDF term. The predicate of triples can only be an IRI. An RDF graph is a set of RDF triples in which nodes are used to represent the subject and object of RDF triples and edges to represent the predicate of triples. RDFS [16] extends RDF with more expressive representation features to allow for modeling schema information such as sub-concept relationships.

OWL [22] and its revision OWL 2 [42] provide a powerful and flexible ontology modeling language that can capture features such as concept hierarchies, incomplete, or negative information. OWL has been historically considered as an extension of RDF, that is why there are two kinds of formal semantics for OWL, i.e., the Direct Semantics [93] and the RDF-based Semantics [116]. The Direct Semantics of OWL is defined for a fragment of OWL, called *OWL DL* that satisfies certain syntactic conditions [94]. OWL Full, i.e., OWL without any restriction, can only be interpreted under the RDF-based Semantics. Under the Direct Semantics of OWL, there is a direct correspondence between DL axioms and OWL axioms [93, 7] even though OWL axioms have a different representation from DL axioms. Please note that concepts are usually called *classes* and (abstract) roles are called *object properties* in OWL. Moreover, OWL defines data properties (called *concrete roles* in DL terminology) and literals. Without loss of generality we do not consider data properties and literals in this thesis; the presented optimizations can easily be transferred to this case.

We generally abbreviate IRIs using prefixes rdf, rdfs, owl, and xsd to refer to the RDF, RDFS, OWL, and XML Schema Datatypes namespaces, respectively. The empty prefix is used for an imaginary example namespace, which we completely omit in DL syntax.

Example 1. *An example of an RDF graph in the form of RDF triples is given next:*

$$:mary \ \text{:takesCourse} \ :maths. \tag{1.14}$$

$$:maths \ \text{rdf:type} \ \text{:Course}. \tag{1.15}$$

$$\text{:takesCourse rdfs:domain :Student}. \tag{1.16}$$

Triple (1.14) *states that mary takes as course maths. Triple* (1.15) *states that maths is a course. Triple* (1.16) *states that every individual that takes a course is a student.*

SPARQL [110] was standardized in 2008 by the World Wide Web Consortium (W3C) and has recently been extended to SPARQL 1.1 [50]. Since 2008, SPARQL has developed into the main query language for the Semantic Web and is now supported by most RDF triple stores. The query criteria are given in the form of RDF triples possibly with variables in place of subject, predicate or object of the triples (this kind of triples with variables are called *triple patterns*). The subject and the object of a triple pattern can be an RDF term or a variable, while the

predicate can be an IRI or a variable. A set of triple patterns is called a *basic graph pattern* (BGP). The query evaluation mechanism defined in the SPARQL Query specification [110] is based on *subgraph matching*. Each instantiation of the variables that yields a subgraph of the queried RDF graph constitutes a solution. This form of query evaluation is also called *simple entailment* since it can equally be defined in terms of the simple entailment relation between RDF graphs [51].

Example 2. *An example of a SPARQL query is*

$$SELECT\ ?x, ?y\ FROM <ontologyIRI>\ WHERE\ \{\ ?x\ \text{:takesCourse}\ ?y.\ \}$$

The FROM clause of the SPARQL query contains the IRI of the queried RDF graph while the WHERE clause (the part of the SPARQL query where the query criteria are stated) consists of a basic graph pattern: an RDF graph written in Turtle syntax [10], where some nodes or edges are replaced by variables. The SELECT clause contains the variables to be selected, i.e., the variables whose bindings will appear in the query's result sequence. The answer to this SPARQL query over the RDF graph of Example 1 is, according to subgraph matching, the mapping $?x \mapsto$:*mary*, $?y \mapsto$:*maths*. Although the SPARQL specification uses Turtle, other query syntaxes can also be defined. Pellet accepts, for example, queries where the BGP is written in Manchester Syntax [56].

Example 3. *If we assume now that we have the following SPARQL query*

$$SELECT\ ?x, ?y\ FROM <ontologyIRI>\ WHERE\ \{\ ?x\ \text{rdf:type}\ \text{:Student}.\ \}$$

the answer to this query over the RDF graph of Example 1 is the empty set under the standard SPARQL semantics, i.e., subgraph matching. This happens because there is no explicitly stated tuple in the queried RDF graph which states that mary is a student. In order to deduce this fact we need RDFS or OWL entailment. SPARQL 1.1 includes several *entailment regimes* [36] in order to use more elaborate entailment relations, such as those induced by RDF Schema (RDFS) [51] or OWL [93, 116]. Query answering under such entailment regimes is more complex as it may involve retrieving answers that only follow implicitly from the queried graph. While several implementations for SPARQL's RDFS entailment regime are available (e.g., Oracle 11g,[1] Apache Jena,[2] or Stardog[3]), the development of tools that provide full SPARQL support under OWL semantics is still an ongoing effort.

In this thesis we are interested in the OWL 2 Direct Semantics entailment regime of SPARQL [36, 31], which uses the OWL 2 Direct Semantics entailment relation [93]. Since the Direct Semantics of OWL is defined in terms of OWL structural objects [94], the first step of the translation process is to map the queried RDF graph to an OWL ontology. Note that a mapping from RDF graphs to OWL ontologies is only defined for certain well-formed RDF graphs that correspond to OWL 2 DL ontologies (ontologies that satisfy certain syntactic conditions) [105]. The RDF graph of Example 1 can be translated to an OWL 2 DL ontology $\mathcal{O}_G$ containing the

[1] `http://www.oracle.com/de/products/database/overview/index.html`
[2] `http://jena.apache.org`
[3] `http://stardog.com`

following structural objects in functional-style syntax (FSS) [94].

$$\mathsf{ObjectPropertyAssertion}(\mathsf{:takesCourse}\ \mathit{:mary}\ \mathit{:maths}) \tag{1.17}$$

$$\mathsf{ClassAssertion}(\mathsf{:Course}\ \mathit{:maths}) \tag{1.18}$$

$$\mathsf{ObjectPropertyDomain}(\mathsf{:takesCourse\ :Student}) \tag{1.19}$$

which in DL notation can be written $\mathsf{takesCourse}(\mathit{mary}, \mathit{maths})$, $\mathsf{Course}(\mathit{maths})$, $\exists \mathsf{takesCourse}.\top \sqsubseteq \mathsf{Student}$. Note that some OWL modeling constructs correspond to several RDF triples and the RDF triples might contain auxiliary blank nodes that are not part of the corresponding OWL object and are not visible in the corresponding axiom in FSS. For example, OWL axioms using the ObjectSomeValuesFrom complex class expression like in

$$\mathsf{ClassAssertion}(\mathsf{ObjectSomeValuesFrom}(\mathsf{:takesCourse\ :GraduateCourse})\ \mathit{:mary})$$

stating that mary takes at least one graduate course is translated to the following RDF triples:

$$\begin{aligned}&\mathit{:mary}\ \mathsf{rdf{:}type}\ \_{:}\mathrm{b}\\ &\_{:}\mathrm{b}\ \mathsf{rdf{:}type\ owl{:}Restriction}\ ;\\ &\_{:}\mathrm{b}\ \mathsf{owl{:}onProperty\ :takesCourse}\ ;\\ &\_{:}\mathrm{b}\ \mathsf{owl{:}someValuesFrom\ :GraduateCourse}\end{aligned}$$

in which the blank node _:b is used to connect the RDF triples. After the translation of the queried RDF graph to an OWL ontology, the BGP(s) of the query are mapped to (extended) structural objects, which can have variables in place of classes, object properties or individuals. According to this, the BGP of Example 3 is mapped to $\mathsf{ClassAssertion}(\mathsf{:Student}\ ?x)$ in functional-style syntax or $\mathsf{Student}(?x)$ in DL syntax. We call such extended DL axioms *axiom templates*. The axiom $\mathsf{Student}(\mathit{mary})$ is entailed by the ontology $\mathcal{O}_G$ under the Direct Semantics of DLs (or OWL). Hence, the mapping $?x \mapsto \mathit{mary}$ is returned as an answer to the query of Example 3 by SPARQL 1.1.

We do not recall the complete surface syntax and semantics of SPARQL here since the only part that is specific to the evaluation of SPARQL queries under OWL's Direct Semantics is the evaluation of BGPs. More complex WHERE clauses, which use operators such as UNION for alternative selection criteria or OPTIONAL to query for optional bindings [110], can be evaluated simply by combining the results obtained by the BGP evaluation. Similarly, the projection of variables from the SELECT clause is a straightforward operation over the results of the evaluation of the WHERE clause. Therefore, we focus here on BGP evaluation only.

Please note that there are two types of individual variables in SPARQL; standard (distinguished) variables and anonymous individuals (also known as blank nodes). The anonymous individuals in the query are treated like distinguished variables with the difference that they cannot be selected and, hence, their bindings cannot appear in the query answer. This is in contrast to conjunctive queries, where anonymous individuals are treated as existential variables. On the other hand, anonymous individuals can occur in the query answer as bindings to distinguished variables, i.e., SPARQL treats anonymous individuals from the queried ontology as (Skolem) constants. This treatment of anonymous individuals has been chosen for compatibility with SPARQL's standard subgraph matching semantics.

In order to implement the RDF(S) entailment regime, systems can simply extend the queried graph with inferred information (materialization) and can then use SPARQL's standard evaluation mechanism over the materialized graph in order to compute the query results. Similarly, when users move on to systems that support the OWL RL profile [92], the OWL RL rule set from the OWL 2 specification can be used to compute the query answers (again via materialization). If one were to change the semantics of blank nodes for SPARQL's entailment regimes to reflect conjunctive query semantics, one could no longer use materialization plus a standard SPARQL query processor to implement the entailment regime. This happens because, intuitively, the presence of an assertion containing a blank node with existential semantics in the queried ontology is equivalent to an assertion axiom with an existential restriction, which is not included in the materialization set (the materialization set contains only assertions about named individuals), i.e., an axiom of the form $\mathsf{r}(a, \_{:}b)$, in which $\_{:}b$ has an existential semantics is equivalent to an axiom of the form $(\exists \mathsf{r}.\top)(a)$. We now explain this point in more detail through an example. Let us assume that we have the RDF triple $:a$ $:\mathsf{r}$ $\_{:}b$ in which the blank node $\_{:}b$ has an existential semantics and the BGP $\{?x$ $:\mathsf{r}$ $?y\}$, in which $?y$ is an existential (non-distinguished) variable. The triple $:a$ $:\mathsf{r}$ $\_{:}b$ is not included in the materialization set since $\_{:}b$ has an existential semantics and hence the answers to the above BGP is the empty set and not the mapping $?x \mapsto :a$. This means that the materialization would not be a sound and complete procedure for query answering in this case.

If one were to change the semantics of blank nodes only for the OWL Direct Semantics entailment regime, where materialization cannot (directly) be used to implement the regime, it could happen that users get less answers for queries posed over an OWL DL ontology than for queries posed over an RDF(S) subset of the OWL DL ontology, which is counter-intuitive. For example, let us assume that we have the RDF triple $:a$ $:\mathsf{r}$ $\_{:}b$ and the BGP $\{?x$ $:\mathsf{r}$ $?y\}$ of a SPARQL query. The answer to this query in RDF(S) under SPARQL's semantics (i.e., if we treat $\_{:}b$ as a constant) would be $?x \mapsto :a$, $?y \mapsto$ $\_{:}b$. Now, if we see the queried graph as an OWL ontology containing the axiom $\mathsf{r}(a, \_{:}b)$, the BGP as the axiom template $\mathsf{r}(?x, ?y)$ and we assume that _:b has an existential semantics (we assume that $?y$, as in the case of RDF(S) is a distinguished variable) then the answer of the same query over the equivalent OWL ontology would be different from the case of RDF(S), i.e., the empty set, because variable $?y$ from the BGP needs to be mapped to a concrete named individual. For brevity and without loss of generality, in the following we assume that neither the query nor the queried ontology contains anonymous individuals.

For a more detailed introduction to SPARQL queries and their algebra we refer interested readers to [53, 106]. For further details about the translation procedure, we refer interested readers to the W3C specification that defines the mapping between RDF graphs and OWL structural objects [105] and to the W3C specification of the OWL Direct Semantics entailment regime of SPARQL [37] that defines the extension of this mapping between BGPs and OWL objects with variables.

The range of queries that can be formulated in SPARQL under the OWL Direct Semantics entailment regime, i.e., SPARQL-OWL, goes beyond standard conjunctive queries [32, 101], which are already supported by several OWL reasoning systems. Although SPARQL-OWL does not allow for proper non-distinguished variables, it poses significant challenges for implementations, since, for example,

variables can occur within complex class expressions and can also bind to class or property names [78, 77, 76].

There is not yet a standardized and commonly implemented query language for expressive OWL DL ontologies. Several of the widely deployed systems support, however, some query language. Pellet supports SPARQL-DL [125], which is a subset of SPARQL, adapted to work with OWL's Direct Semantics. The kinds of SPARQL queries that are supported in SPARQL-DL are those that can directly be mapped to reasoner tasks. Similarly, KAON2 [65, 64] supports SPARQL queries, but restricted to ABox queries/conjunctive instance queries. Racer Pro [47] has a proprietary query language, called nRQL [49], which allows for queries that go beyond ABox queries, e.g., one can retrieve sub- or super-concepts of a given concept. TrOWL[4] is another system that supports SPARQL queries, but the reasoning in TrOWL is approximate, i.e., an OWL DL ontology is rewritten into an ontology that uses a less expressive language before reasoning is applied [134]. Furthermore, there are systems such as QuOnto[5] or Requiem,[6] which support profiles of OWL 2, and which support conjunctive queries, e.g., written in SPARQL syntax, but with proper non-distinguished variables. Of the systems that support all of OWL 2 DL, only Pellet supports non-distinguished variables as long as they are not used in cycles, since decidability of cyclic conjunctive queries is to the best of our knowledge still an open problem.

1.3 Thesis Contribution

Over the last decade, much effort has been spent on optimizing standard reasoning tasks such as entailment checking, classification (i.e., the computation of the concept hierarchy), or realization (i.e., the computation of instances of all concepts and roles) [7, 57, 123, 137, 33]. The optimization of query answering algorithms has, however, mostly been addressed for conjunctive queries in OWL profiles, most notably the OWL 2 QL profile [18, 82, 108, 112].

In the current thesis we address the problem of efficient SPARQL query evaluation over OWL 2 DL/$\mathcal{SROIQ}$ ontologies by proposing a query answering algorithm and a range of novel optimizations that deal in particular with the expressive features of SPARQL such as variables in place of concepts or roles. To the best of our knowledge, there is no other work yet on the optimization of such complex queries with variables in nested positions in axiom templates.

Since consistency checking is an expensive knowledge base operation (for the case of $\mathcal{SROIQ}$ it is 2-NExpTime complete) to which every other standard reasoning task is reduced, as we explained in Section 1.1, our goal in the thesis is to reduce the number of consistency checks that are performed in the process of answering a SPARQL-OWL query, which, in turn, generally leads to a reduction in the query execution time. In order to achieve this, we first present a general query answering algorithm and we afterwards work in two directions. On the one hand, we adapt the cost-based query planning techniques, commonly used in the area of databases [2, 29], to work with ontological knowledge and with complex queries that can be

[4] http://trowl.eu

[5] http://www.dis.uniroma1.it/~quonto/

[6] http://www.comlab.ox.ac.uk/projects/requiem/home.html

expressed in SPARQL-OWL. On the other hand, we propose query rewriting techniques and show how specialized OWL reasoning tasks and the concept and role hierarchies can be used to reduce the number of performed consistency checks and thus the query execution time for complex queries.

The proposed approach has been implemented and a system has been developed for efficiently answering SPARQL-OWL queries.[7] Using this tool, we evaluate the efficiency of the proposed query ordering and optimization techniques.

1.4 Thesis Outline

The thesis is organized as follows:

- In Chapter 2 we present the standard syntax and Tarski-style model theoretic semantics of Description Logics and the reasoning tasks that they support. Moreover, the syntax and semantics of the SPARQL-OWL queries are formalized.
- In Chapter 3 we present some widely used query answering and optimization approaches in the area of databases and ontologies and explain how these techniques are related or differ from our approach. For query answering techniques over less expressive DLs than $\mathcal{SROIQ}$; i.e., the DL that is considered in the current thesis, we focus on possible problems one faces if one tries to extend these techniques to more expressive DLs.
- In Chapter 4 we describe a general query evaluation algorithm and present the foundations of our cost model regarding the ordering of query atoms. This includes a description of the information that is extracted from the reasoner model construction procedure and an analysis of how this information is used to define the cost functions.
- In Chapter 5 we propose optimizations for complex axiom templates, i.e., axiom templates in which variables appear in nested concept or role positions. The notion of the polarity of variables within axiom templates is defined, which is exploited to reduce the performed consistency checks through the pruning of possible query mappings.
- In Chapter 6 we present the proposed query answering algorithm which exploits the optimizations developed in Chapters 4 and 5 and we prove termination, soundness and completeness of the algorithm.
- In Chapter 7 an evaluation showing the effectiveness of the proposed optimized query answering approach is presented.
- In Chapter 8 we show how the implemented system combined with a query rewriting system can be used on a cultural heritage application scenario.
- In Chapter 9 the contribution of the thesis is summarized and directions are given for future work.

The work in this thesis has been published in several conferences and journals [73, 74, 81, 80, 79, 75, 35].

[7] `https://code.google.com/p/owl-bgp/`

Chapter 2

Foundations of Description Logics

In this chapter we give a formal introduction in Description Logics (DLs) (Section 2.1) and in the reasoning techniques used in DLs (Section 2.2). Moreover, we describe a Description Logic query language (Section 2.3), the syntax and semantics of which is based on the OWL Direct Semantics entailment regime of SPARQL as discussed in Section 1.2.

2.1 Syntax and Semantics

$\mathcal{SROIQ}$ is the description logic underpinning OWL 2 DL which is the language considered in the current thesis. The optimizations we present do not need all features of $\mathcal{SROIQ}$, hence we here only present $\mathcal{SHOIQ}$ formally, which allows for a shorter and easier to follow presentation and we afterwards discuss the additional features of $\mathcal{SROIQ}$. In the following, we usually refer to knowledge bases as ontologies. We now define the syntax and semantics of $\mathcal{SHOIQ}$ as well as the most widely used DL reasoning services.

Definition 1 (**Syntax of $\mathcal{SHOIQ}$).** *Let N_C, N_R, and N_I be countable, infinite, and pairwise disjoint sets of* concept names, role names, *and* individual names, *respectively. We call $\mathcal{S} = (N_C, N_R, N_I)$ a* signature. *The set $\mathsf{rol}(\mathcal{S})$ of $\mathcal{SHOIQ}$-*roles *over $\mathcal{S}$ (or roles for short) is $N_R \cup \{r^- \mid r \in N_R\}$, where roles of the form r^- are called* inverse roles. *A* role inclusion axiom *is of the form $r \sqsubseteq s$ with r, s roles. A* transitivity axiom *is of the form $\mathsf{trans}(r)$ for r a role. A* role hierarchy *$\mathcal{H}$ is a finite set of role inclusion and transitivity axioms.*

For a role hierarchy $\mathcal{H}$, we define the function inv *over roles as $\mathsf{inv}(r) := r^-$ if $r \in N_R$ and $\mathsf{inv}(r) := s$ if $r = s^-$ for a role name $s \in N_R$. Further, we define $\sqsubseteq_\mathcal{H}$ as the smallest transitive reflexive relation on roles such that $r \sqsubseteq s \in \mathcal{H}$ implies $r \sqsubseteq_\mathcal{H} s$ and $\mathsf{inv}(r) \sqsubseteq_\mathcal{H} \mathsf{inv}(s)$. We write $r \equiv_\mathcal{H} s$ if $r \sqsubseteq_\mathcal{H} s$ and $s \sqsubseteq_\mathcal{H} r$. A role r is* transitive w.r.t. *$\mathcal{H}$ (notation $r^+ \sqsubseteq_\mathcal{H} r$) if a role s exists such that $r \sqsubseteq_\mathcal{H} s$, $s \sqsubseteq_\mathcal{H} r$, and $\mathsf{trans}(s) \in \mathcal{H}$ or $\mathsf{trans}(\mathsf{inv}(s)) \in \mathcal{H}$. A role s is called* simple w.r.t. *$\mathcal{H}$ if there is no role r such that r is transitive w.r.t. $\mathcal{H}$ and $r \sqsubseteq_\mathcal{H} s$.*

*Given a signature $\mathcal{S} = (N_C, N_R, N_I)$ and a role hierarchy $\mathcal{H}$, the set of $\mathcal{SHOIQ}$-*concepts *(or concepts for short) over $\mathcal{S}$ is the smallest set built inductively over symbols from $\mathcal{S}$ using the following grammar, where $o \in N_I, A \in N_C, n \in \mathbb{N}_0$, s is a simple role w.r.t. $\mathcal{H}$, and r is a role w.r.t. $\mathcal{H}$:*

$$C ::= \top \mid \bot \mid \{o\} \mid A \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \forall r.C \mid \exists r.C \mid \leqslant n\, s.C \mid \geqslant n\, s.C.$$

In number restrictions only simple roles are allowed and this is a measure to ensure decidability of the Description Logic. In the following, we use $= n\ s.C$ as a syntactic shortcut for $\leqslant n\ s.C$ and $\geqslant n\ s.C$.

Definition 2 (**Semantics of $\mathcal{SHOIQ}$-concepts**). *An* interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *consists of a non-empty set $\Delta^{\mathcal{I}}$, the* domain *of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$, which maps every concept name $A \in N_C$ to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, every role name $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every individual name $a \in N_I$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. For each role name $r \in N_R$, the interpretation of its inverse role $(r^-)^{\mathcal{I}}$ consists of all pairs $\langle \delta, \delta' \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for which $\langle \delta', \delta \rangle \in r^{\mathcal{I}}$.*

The semantics of $\mathcal{SHOIQ}$-concepts over a signature $\mathcal{S}$ is defined as follows:

$$
\begin{array}{rrr}
\top^{\mathcal{I}} = \Delta^{\mathcal{I}} & \bot^{\mathcal{I}} = \emptyset & (\{o\})^{\mathcal{I}} = \{o^{\mathcal{I}}\} \\
(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} & (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} & (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall r.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \text{if } \langle \delta, \delta' \rangle \in r^{\mathcal{I}}, \text{ then } \delta' \in C^{\mathcal{I}}\} & & \\
(\exists r.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \text{there is a } \langle \delta, \delta' \rangle \in r^{\mathcal{I}} \text{ with } \delta' \in C^{\mathcal{I}}\} & & \\
(\leqslant n\ s.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \sharp(s^{\mathcal{I}}(\delta, C)) \leq n\} & & \\
(\geqslant n\ s.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \sharp(s^{\mathcal{I}}(\delta, C)) \geq n\} & &
\end{array}
$$

where $\sharp(M)$ denotes the cardinality of the set M and $s^{\mathcal{I}}(\delta, C)$ is defined as

$$\{\delta' \in \Delta^{\mathcal{I}} \mid \langle \delta, \delta' \rangle \in s^{\mathcal{I}} \text{ and } \delta' \in C^{\mathcal{I}}\}.$$

Definition 3 (**Syntax and Semantics of Axioms and Ontologies, Entailment**). *For C, D concepts, a* (general) concept inclusion axiom *(GCI) is an expression $C \sqsubseteq D$. We introduce $C \equiv D$ as an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$. A finite set of GCIs is called a* TBox. *A* (concept or role) assertion axiom *is an expression of the form $C(a)$, $r(a, b)$, $\neg r(a, b)$, $a \approx b$, or $a \not\approx b$, where C is a concept, r is a role, and $a, b \in N_I$ are individual names. An* ABox *is a finite set of assertion axioms. An* ontology *$\mathcal{O}$ is a triple $(\mathcal{T}, \mathcal{H}, \mathcal{A})$ with $\mathcal{T}$ a TBox, $\mathcal{H}$ a role hierarchy, and $\mathcal{A}$ an ABox. We use $N_C^{\mathcal{O}}$, $N_R^{\mathcal{O}}$, and $N_I^{\mathcal{O}}$ to denote, respectively, the set of concept, role, and individual names occurring in $\mathcal{O}$.*

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation. Then $\mathcal{I}$ satisfies *a role inclusion axiom $r \sqsubseteq s$ if $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$, $\mathcal{I}$ satisfies a transitivity axiom* $\mathsf{trans}(r)$ *if $r^{\mathcal{I}}$ is a transitive binary relation, and a role hierarchy $\mathcal{H}$ if it satisfies all role inclusion and transitivity axioms in $\mathcal{H}$. The interpretation $\mathcal{I}$* satisfies *a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; and $\mathcal{I}$* satisfies *a TBox $\mathcal{T}$ if it satisfies each GCI in $\mathcal{T}$. The interpretation $\mathcal{I}$* satisfies *an assertion axiom $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, $r(a, b)$ if $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$, $\neg r(a, b)$ if $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \notin r^{\mathcal{I}}$, $a \approx b$ if $a^{\mathcal{I}} = b^{\mathcal{I}}$, and $a \not\approx b$ if $a^{\mathcal{I}} \neq b^{\mathcal{I}}$; $\mathcal{I}$* satisfies *an ABox $\mathcal{A}$ if it satisfies each assertion in $\mathcal{A}$. We say that $\mathcal{I}$* satisfies *$\mathcal{O}$ if $\mathcal{I}$ satisfies $\mathcal{T}$, $\mathcal{H}$, and $\mathcal{A}$. In this case, we say that $\mathcal{I}$ is a* model *of $\mathcal{O}$ and write $\mathcal{I} \models \mathcal{O}$. We say that $\mathcal{O}$* is consistent *if $\mathcal{O}$ has a model.*

Given an axiom α, we say that $\mathcal{O}$ entails *α (written $\mathcal{O} \models \alpha$) if every model $\mathcal{I}$ of $\mathcal{O}$ satisfies α.*

Note that the axiom $\neg r(a, b)$ is only allowed in $\mathcal{SROIQ}$, but it is equivalent to the axiom $\{a\} \sqsubseteq \forall r.\neg\{b\}$, or equivalently $(\forall r.\neg\{b\})(a)$ that are both allowed in $\mathcal{SHOIQ}$, which means that the addition of it in $\mathcal{SROIQ}$ is syntactic sugar. The description logic $\mathcal{SROIQ}$ further allows for a number of features:

- It allows role chains (complex role inclusion axioms) of the form $\mathsf{hasFather} \circ \mathsf{hasBrother} \sqsubseteq \mathsf{hasUncle}$. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies a role chain $r_1 \circ r_2 \sqsubseteq r_3$ if for every $\delta, \delta', \delta'' \in \Delta^{\mathcal{I}}$, the following holds: if $\langle \delta, \delta' \rangle \in r_1^{\mathcal{I}}$ and $\langle \delta', \delta'' \rangle \in r_2^{\mathcal{I}}$ then $\langle \delta, \delta'' \rangle \in r_3^{\mathcal{I}}$.

- It supports the special concept Self, which can be used in axioms of the form $\mathsf{Narcissist} \sqsubseteq \exists \mathsf{loves}.\mathsf{Self}$. For $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ an interpretation and $\delta \in \Delta^{\mathcal{I}}$ it holds $(\exists r.\mathsf{Self})^{\mathcal{I}} = \{\delta \mid \langle \delta, \delta \rangle \in r^{\mathcal{I}}\}$.

- It allows for defining roles that are reflexive ($\mathsf{Ref}(r)$), irreflexive ($\mathsf{Irr}(r)$), symmetric ($\mathsf{Sym}(r)$), asymmetric ($\mathsf{Asy}(r)$) and it allows disjoint roles ($\mathsf{Dis}(r_1, r_2)$). Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation and $\delta, \delta', \delta'' \in \Delta^{\mathcal{I}}$. Then $\mathcal{I}$ satisfies:
  - $\mathsf{Ref}(r)$ if $\langle \delta, \delta \rangle \in r^{\mathcal{I}}$ for every $\delta \in \Delta^{\mathcal{I}}$
  - $\mathsf{Irr}(r)$ if $\langle \delta, \delta \rangle \notin r^{\mathcal{I}}$ for every $\delta \in \Delta^{\mathcal{I}}$
  - $\mathsf{Sym}(r)$ if $\langle \delta, \delta' \rangle \in r^{\mathcal{I}}$ implies $\langle \delta', \delta \rangle \in r^{\mathcal{I}}$
  - $\mathsf{Asy}(r)$ if $\langle \delta, \delta' \rangle \in r^{\mathcal{I}}$ implies $\langle \delta', \delta \rangle \notin r^{\mathcal{I}}$
  - $\mathsf{Dis}(r_1, r_2)$ if $r_1^{\mathcal{I}} \cap r_2^{\mathcal{I}} = \emptyset$

- It allows for defining the top or universal role ($\top_r$). Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation and $\delta, \delta' \in \Delta^{\mathcal{I}}$. $\top_r$ is interpreted as $\{\langle \delta, \delta' \rangle \mid \delta, \delta' \in \Delta^{\mathcal{I}}\}$.

To avoid the undecidability result that comes from the addition of complex role inclusion axioms to an ontology, that can be as simple as $\mathcal{ALC}$, a syntactic regularity condition is placed on role chains. Moreover, restrictions are placed on roles that can be used in some of the additional axioms; for example only simple roles are allowed in axioms of the form $\mathsf{Ref}(r)$, $\mathsf{Irr}(r)$, $\mathsf{Sym}(r)$, $\mathsf{Asy}(r)$, $\mathsf{Dis}(r_1, r_2)$ and in number restrictions (as in $\mathcal{SHOIQ}$). For a detailed introduction to $\mathcal{SROIQ}$ we refer interested readers to [58]. In the following, we assume that the set of role names N_R contains the top role, $\top_r$, and the bottom role, $\bot_r$, which is interpreted as $\emptyset$. Moreover, we assume that the sets $N_C^{\mathcal{O}}$ and $N_R^{\mathcal{O}}$ for an ontology $\mathcal{O}$ contain the concepts $\top$, $\bot$ and the roles $\top_r$ and $\bot_r$ respectively, irrespective of whether these appear in $\mathcal{O}$. Furthermore, we assume that the role hierarchy $\mathcal{H}$ of an ontology is integrated in the TBox $\mathcal{T}$ and, hence, when we use $\mathcal{T}$, we actually mean $\mathcal{T} \cup \mathcal{H}$.

We next define the standard reasoning tasks with respect to an ontology $\mathcal{O}$. Note that all these reasoning problems can be reduced to ontology consistency checking.

Definition 4 (**Standard Reasoning Tasks**). Consistency checking *means to check whether a given ontology $\mathcal{O}$ is* consistent*, i.e., whether it has a model.* Concept satisfiability checking *means to check, given an ontology $\mathcal{O}$ and a concept C, whether C is* satisifiable *w.r.t. $\mathcal{O}$, i.e., if there is a model $\mathcal{I}$ of $\mathcal{O}$ with $C^{\mathcal{I}} \neq \emptyset$.* Concept subsumption checking *means to check, given an ontology $\mathcal{O}$ and two concepts C and D, whether D* subsumes *C w.r.t. $\mathcal{O}$ (written $\mathcal{O} \models C \sqsubseteq D$), i.e., if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for every model $\mathcal{I}$ of $\mathcal{O}$.* Concept instance checking *means to check, given an ontology $\mathcal{O}$, a concept C and an individual a, whether a is an instance of C w.r.t. $\mathcal{O}$ (written $\mathcal{O} \models C(a)$), i.e., if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{O}$.* Role instance checking *means to check, given an ontology $\mathcal{O}$, a role r, and two individuals a and b, whether the pair of individuals $\langle a, b \rangle$ is an instance of the role r w.r.t. $\mathcal{O}$ (written $\mathcal{O} \models r(a, b)$), i.e., if $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{O}$.*

2.1.1 Expressive Description Logics - The DL-Lite Family

As we discussed in Section 1.1, the expressivity of a DL depends on the constructors that are used. For example, the description logic $\mathcal{ALC}$ allows the constructors $\top, \bot, \neg, \sqcap, \sqcup, \exists, \forall$. The extension of $\mathcal{ALC}$ with transitive roles, i.e., $\mathcal{ALC}_{R^+}$ is usually denoted by the letter $\mathcal{S}$. The use of additional letters denotes an additional constructor, for example the letter $\mathcal{I}$ is used for inverse roles, the letter $\mathcal{Q}$ for number restrictions, the letter $\mathcal{O}$ for nominals, the letter $\mathcal{H}$ for role inclusions and the letter $\mathcal{R}$ for complex role inclusions, reflexivity and role disjointness. This naming scheme explains the name of $\mathcal{SROIQ}$. Moreover, each logic can further be extended with concrete domains, such as strings or integers. In such a case, one distinguishes between *abstract roles* that relate two individuals and *concrete roles* that relate an individual with a data value, as discussed in Section 1.2. The extension of a DL with concrete domains is denoted by the letter $\mathcal{D}$ in parenthesis after the DL name.

2.1.2 Light-weight Description Logics

There are applications, in which the need for efficient and scalable reasoning dominates over the need for expressive knowledge representation formalisms, the reasoning tasks in which have a high computational complexity. For this reason, three light-weight Description Logics, each a fragment of $\mathcal{ALC}$, have been introduced that have favorable computational properties and which correspond to fragments of OWL. These are DLP, i.e., a shortcut used for Description Logic Programs, DL-Lite and $\mathcal{EL}$. The most important reasoning tasks are tractable in each of these languages, i.e., they can be performed in polynomial time. For example, these logics allow for tractable query answering in data complexity [104]. In particular, DLP [44], is a fragment containing axioms that can be translated to rules in First Order Horn Logic without function symbols. It corresponds to the OWL 2 RL fragment of OWL and it is widely used for reasoning with Web data. $\mathcal{EL}$ and $\mathcal{EL}^{++}$ [5] underpinning OWL 2 EL is used in huge biomedical ontologies that consist mainly of terminological axioms. DL-Lite [18] is widely used in applications where there is a huge amount of data stored in traditional databases. Query answering in DL-Lite can be realized through the use of a query rewriting approach and the translation of rewritten conjunctive queries to SQL queries, which are afterwards evaluated over the database with traditional database query answering techniques. We now present the axioms that are allowed in the basic DL-Lite, i.e., DL-Lite$_{\text{core}}$, so that the reader can get an idea of the expressive power of DL-Lite. DL-Lite$_{\text{core}}$ only allows concept inclusion axioms of the form $B_1 \sqsubseteq B_2$, where B_1 is a concept of the form A or $\exists r.\top$, or $\exists r^-.\top$ for some concept name $A \in N_C$ and some role name $r \in N_R$, while B_2 is a possibly negated concept of the same form as B_1, i.e., B_2 is of the form A, $\exists r.\top$, $\exists r^-.\top$, $\neg A$, $\neg\exists r.\top$ or $\neg\exists r^-.\top$. An extension of DL-Lite$_{\text{core}}$, called DL-Lite$_R$, which provides the logical underpinning for OWL 2 QL additionally allows for role hierarchies.

2.2 Reasoning Techniques

In this section, we give a brief overview over the main reasoning techniques used in $\mathcal{SROIQ}$ ontologies since our cost-based query planning and the other presented

optimizations rely on these techniques.

In order to check whether an ontology $\mathcal{O}$ entails an assertion axiom α, one typically checks whether $\mathcal{O} \cup \{\neg\alpha\}$ has a model. If that is not the case, then every model of $\mathcal{O}$ satisfies α and $\mathcal{O} \models \alpha$. For example, to check whether an individual a is an instance of a concept C w.r.t. an ontology $\mathcal{O}$, we check whether adding the concept assertion $\neg C(a)$ to $\mathcal{O}$ leads to an inconsistency. To check this, most OWL reasoners use a model construction calculus such as tableau or hypertableau. (Hyper)tableau calculi [96] start from the initial set of ABox assertions and, by applying derivation rules, they try to construct (an abstraction of) a model of $\mathcal{O}$. Derivation rules usually add new concept and role assertion axioms, they may introduce new individuals, they can be non-deterministic, leading to the need to choose between several alternative assertions to add or they can lead to a *clash* when a contradiction is detected. To show that an ontology $\mathcal{O}$ is (in)consistent, (hyper)tableau calculi construct a *derivation*, i.e., a sequence of sets of assertions $\mathcal{A}_0, \ldots, \mathcal{A}_n$, such that

- $\mathcal{A}_0$ contains all ABox assertions in $\mathcal{O}$,
- $\mathcal{A}_{i+1}$ is the result of applying a derivation rule to $\mathcal{A}_i$ and
- $\mathcal{A}_n$ is the final set of assertions where no more rules are applicable.

If a derivation exists such that $\mathcal{A}_n$ does not contain a clash, then $\mathcal{O}$ is consistent and $\mathcal{A}_n$ is called a *pre-model* of $\mathcal{O}$. Otherwise $\mathcal{O}$ is inconsistent. Each assertion in a set of assertions $\mathcal{A}_i$ is derived either deterministically or non-deterministically. An assertion is *derived deterministically* if it is derived by the application of a deterministic derivation rule from assertions that were all derived deterministically. Any other derived assertion is *derived non-deterministically*. It is easy to know whether an assertion was derived deterministically or not because of the dependency directed backtracking [7, 137] that most (hyper)tableau reasoners employ. In the pre-model, each individual a is assigned a label $\mathcal{L}(a)$ representing the concepts it is (non)deterministically an instance of and each pair of individuals $\langle a, b\rangle$ is assigned a label $\mathcal{L}(\langle a, b\rangle)$ representing the roles through which individual a is (non)deterministically related to individual b. Given a pre-model, a model for $\mathcal{O}$ can be constructed by a process called *unraveling* [139].

We next briefly describe tableau calculi together with optimizations pointing to their differences from hypertableau calculi [96], which we use in the current thesis. In tableau algorithms, the commonly used derivation rules are shown below:

- $\sqcup$-rule: Given $(C_1 \sqcup C_2)(a)$, derive either $C_1(a)$ or $C_2(a)$
- $\sqcap$-rule: Given $(C_1 \sqcap C_2)(a)$, derive $C_1(a)$ and $C_2(a)$
- $\exists$-rule: Given $\exists r.C(a)$, derive $r(a, b)$ and $C(b)$ for b a fresh individual
- $\forall$-rule: Given $\forall r.C(a)$ and $r(a, b)$, derive $C(b)$
- $\sqsubseteq$-rule: Given a GCI $C \sqsubseteq D$ and an individual a, derive $(\neg C \sqcup D)(a)$

The $\sqcup$-rule and $\sqsubseteq$-rule are non-deterministic in the sense that, when they are invoked, tableau algorithms need to make a non-determinsitc guess and backtrack in case this guess leads to a contradiction, something which can give rise to exponential behaviour. The $\sqsubseteq$-rule, in particular, adds a disjunction for each GCI to each

individual in an ABox and therefore is a major source of inefficiency. To control the non-determinism arising from GCIs most tableau calculi employ *absorption* optimizations [7, 135, 62]. The basic absorption algorithm tries to rewrite GCIs into the form $A \sqsubseteq C$ where A is a concept name. After such preprocessing, instead of deriving $\neg A \sqcup C$ for each individual in an ABox, $C(a)$ is derived only if an ABox contains $A(a)$. In this way the non-determinism introduced by the absorbed GCIs is localized. Heuristics are often used to find good absorptions (i.e., absorptions that lead to a small amount of non-determinism), however, it is not guaranteed that the optimal absorption is found. The used hypertableau calculus generalizes all known absorption techniques and guarantees deterministic behaviour when the input ontology is Horn[1] and the least amount of non-determinism otherwise. Naive application of (hyper)tableau rules does not always terminate. For example, consider a TBox containing the axiom $B \sqsubseteq \exists r.B$ and the assertion $B(a)$. It is obvious that, in this case, the axioms are only satisfied in an infinite model and tableau algorithms would construct an infinite number of ABoxes during the model construction phase, which contain a chain of r-successors. To ensure termination (i.e., to ensure that only finite model abstractions are constructed) in such cases, (hyper)tableau algorithms employ *blocking*. Only the (new) individuals created due to existential restrictions can be blocked and thus these individuals are called *blockable*, in contrast to *named* individuals appearing in the ABox. Note that named individuals can be connected in an arbitrary way, whereas blockable individuals can only be connected in a forest shaped way (assuming that the DL under consideration does not contain nominals). Depending on the expressivity of the DL language used, one can employ different kinds of blocking strategies such as subset blocking [6], equality blocking or (ancestor) pairwise blocking [59, 61, 7].

In the remainder we focus on the description of the hypertableau calculus, which is used in the current thesis. The hypertableau algorithm is a hybrid of resolution and tableau. For ease of presentation, we analyze the hypertableau calculus for the description logic $\mathcal{S}$, i.e., $\mathcal{ALC}$ extended with transitive roles. For a more detailed description of hypertableau calculus for more expressive description logics interested readers are referred to [95, 96]. The hypertableau calculus does not operate on $\mathcal{O}$ directly but, in order to reduce non-determinism, it translates a DL knowledge base into a set of DL-clauses $\mathcal{C}$, i.e., clauses of a specific form, and an ABox $\mathcal{A}$ that together are equisatisfiable with $\mathcal{O}$.

Hypertableau (and tableau reasoners) typically do not deal with transitivity directly. Tableau calculi have a specific rule, which can handle concepts of the form $\forall r.C$ where r is a non-simple role. Hypertableau algorithms, on the other hand, translate concepts of the form $\forall r.C$ into DL-clauses, so the tableau rule that handles transitivity cannot be used. That is why hypertableau algorithms perform a preprocessing step which eliminates transitivity axioms from $\mathcal{O}$ but simulates their effect using additional GCIs. More details about this transitivity elimination is given in Section 4.2. It should be noted that, during this preprocessing step, the DL axioms are normalized so that all negations are made explicit and to ensure that the resulting DL-clauses are compatible with blocking.

A DL-clause is a universally quantified implication of the form $\bigwedge_{i=1}^{m} U_i \rightarrow \bigvee_{j=1}^{n} V_j$, where U_i and V_j are called the *antecedent* and the *consequent* atoms respectively.

[1] A Horn ontology is an ontology which contains only Horn clauses, i.e., clauses (disjunctions of literals) that contain at most one positive literal.

Each *antecedent* atom is of the form $A(x), r(x, y)$ and each *consequent* atom is of the form $B(x), \exists r.B(x)$, where A is a concept name, B a possibly negated concept name and r a role. DL-clauses can straightforwardly be interpreted in First Order Logic and, intuitively, they state that at least one consequent atom must be true whenever all atoms in the antecedent are true.

The main derivation rule of the calculus is the Hyp-rule. The Hyp-rule is applicable to a DL-clause $r = \bigwedge_{i=1}^{m} U_i \rightarrow \bigvee_{j=1}^{n} V_j$ and an ABox $\mathcal{A}$ if a mapping σ from the variables in r to the individuals in $\mathcal{A}$ exists such that $\sigma(U_i) \in \mathcal{A}$ for each $1 \leq i \leq m$ and $\sigma(V_j) \notin \mathcal{A}$ for each $1 \leq j \leq n$. In this case, $\mathcal{A}_1 = \mathcal{A} \cup \{\bot\}$ if $n = 0$, otherwise the rule non-deterministically derives $\mathcal{A}_j = \mathcal{A} \cup \{\sigma(V_j)\}$ for $1 \leq j \leq n$.

The $\exists$-rule is applicable to $(\exists r.B)(a) \in \mathcal{A}$ if no individual b exists such that $r(a, b) \in \mathcal{A}$ and $B(b) \in \mathcal{A}$. In this case, $\mathcal{A}$ is extended to $\mathcal{A}_1 = \mathcal{A} \cup \{r(a, b), B(b)\}$, where b is a fresh successor of a.

The $\bot$-rule derives obvious contradictions. If $\mathcal{A}$ contains both $A(a)$ and $\neg A(a)$, the rule derives $\mathcal{A}_1 = \mathcal{A} \cup \{\bot\}$, i.e., $\mathcal{A}_1$ contains a *clash*.

2.3 Ontological Query Answering

Inspired by the SPARQL OWL Direct Semantics entailment regime we define the syntax and semantics of our queries. The BGPs of SPARQL queries can easily be transformed to these DL queries because of the mapping of (sets of) RDF triples to OWL structural objects as discussed in Section 1.2 and the direct translation between OWL structural objects and DL axioms. The optimization techniques that we develop in the current thesis concern the evaluation of each BGP separately, therefore we focus on the BGP parts of SPARQL. In the following, we directly write BGPs in DL notation extended to allow for variables in place of concept, role and individual names in axioms. It is worth reminding that SPARQL does not support existentially quantified variables, which is in contrast to database-style conjunctive queries, where one typically also has existential/non-distinguished variables. For brevity and without loss of generality, we assume here that neither the query nor the queried ontology contains anonymous individuals.

Definition 5 (Query). *Let $\mathcal{S} = (N_C, N_R, N_I)$ be a signature. A* query signature *$\mathcal{S}_q$ w.r.t. $\mathcal{S}$ is a six-tuple $(N_C, N_R, N_I, V_C, V_R, V_I)$, where V_C, V_R, and V_I are countable, infinite, and pairwise disjoint sets of* concept variables*,* role variables*, and* individual variables *disjoint from N_C, N_R, and N_I. A concept term is an element from $N_C \cup V_C$. A role term is an element from $N_R \cup V_R$. An individual term is an element from $N_I \cup V_I$. An* axiom template *over $\mathcal{S}_q$ is a $\mathcal{SROIQ}$ axiom over $\mathcal{S}$, where one can also use concept variables from V_C in place of concept names, role variables from V_R in place of role names, and individual variables from V_I in place of individual names. A* query *q w.r.t. a query signature $\mathcal{S}_q$ is a non-empty set of axiom templates over $\mathcal{S}_q$. We use* $\mathsf{Var}(q)$ ($\mathsf{Var}(\mathsf{at})$ *for an axiom template* at) *to denote the set of all variables in q* (at) *and $|q|$ to denote the number of axiom templates in q. Let t, t' be individual terms; we call axiom templates of the form $A(t)$ with $A \in N_C$* concepts atoms*, templates of the form $r(t, t')$ with $r \in N_R$* role atoms*, and templates of the form $t \approx t'$* equality atoms*; in general we call such templates* query atoms*. A* conjunctive instance query *q w.r.t. a query signature $\mathcal{S}_q$ is a non-empty set of query atoms.*

For a function μ, we use $dom(\mu)$ to denote the domain *of μ. Let $\mathcal{O}$ be an ontology over $\mathcal{S}$ and $q = \{\mathsf{at}_1, \ldots, \mathsf{at}_n\}$ a query over $\mathcal{S}_q$ consisting of n axiom templates. A* mapping μ for q over $\mathcal{O}$ *is a total function $\mu\colon \mathsf{Var}(q) \to N_C^{\mathcal{O}} \cup N_R^{\mathcal{O}} \cup N_I^{\mathcal{O}}$ such that*

1. *$\mu(v) \in N_C^{\mathcal{O}}$ for each $v \in V_C \cap dom(\mu)$,*
2. *$\mu(v) \in N_R^{\mathcal{O}}$ for each $v \in V_R \cap dom(\mu)$,*
3. *$\mu(v) \in N_I^{\mathcal{O}}$ for each $v \in V_I \cap dom(\mu)$, and*
4. *$\mathcal{O} \cup \mu(q)$ is a $\mathcal{SROIQ}$ ontology.*

We write $\mu(q)$ ($\mu(\mathsf{at})$) to denote the result of replacing each variable v in q (at) with $\mu(v)$. The set $\Gamma_q^{\mathcal{O}} := \{\mu \mid \mu$ is a mapping for q over $\mathcal{O}\}$ contains the compatible *mappings for q over $\mathcal{O}$. A mapping μ is a* solution mapping *or a* certain answer *for q over $\mathcal{O}$ if $\mathcal{O} \models \mu(q)$. We denote the set containing all solution mappings for q over $\mathcal{O}$ with $\Omega_q^{\mathcal{O}}$ or with $\mathsf{ans}(\mathcal{O}, q)$. The* result size *or the* number of answers *of a query q over $\mathcal{O}$ is given by the cardinality of the set $\Omega_q^{\mathcal{O}}$ ($\mathsf{ans}(\mathcal{O}, q)$).*

Note that the last condition in the definition of mappings (Condition 4) is required to ensure decidability of query entailment. For example, without this condition, a reasoner might have to test instantiated axiom templates where a role variable has been replaced by a non-simple role in a number restriction, which is not allowed in Description Logic axioms. Note also that in the query we do not indicate which variables are to be selected since we do not consider the straightforward task of projection here. In the following, we sometimes write a mapping μ for a query q over an ontology $\mathcal{O}$ as a set of variable mappings, each element of which maps a variable $?x$ in $\mathrm{dom}(\mu)$ to $\mu(?x)$. We now define the notion of the union of mappings and the notion of the restriction of a mapping over a set of variables, which are used in the next chapters.

Definition 6 (**Union of Mappings**). *Let $\mathcal{O}$ be an ontology, q a query and μ_1, μ_2 mappings for q over $\mathcal{O}$ such that $\mu_1(?x) = \mu_2(?x)$ for all $?x \in dom(\mu_1) \cap dom(\mu_2)$. The union of the mappings μ_1 and μ_2 is defined as follows: $(\mu_1 \cup \mu_2)(?x) = \mu_1(?x)$ if $?x \in dom(\mu_1)$ and $(\mu_1 \cup \mu_2)(?x) = \mu_2(?x)$ otherwise.*

Definition 7 (**Restriction of Mapping over Variables**). *Let $\mathcal{O}$ be an ontology, q a query, μ a mapping for q over $\mathcal{O}$ and $X \subseteq dom(\mu)$ a subset of the variables in the domain of μ. The restriction of μ over X is the mapping $\mu_{|X} = \{?x \mapsto a \mid ?x \in X$ and $\mu(?x) = a\}$.*

In the remainder, unless otherwise stated, we use $\mathcal{S}$ for a signature (N_C, N_R, N_I), $\mathcal{O}$ to denote a $\mathcal{SROIQ}$ ontology over $\mathcal{S}$, $A, B \in N_C$ for concept names from $\mathcal{O}$, $r, s \in N_R$ for role names from $\mathcal{O}$, $a, b \in N_I$ for individual names from $\mathcal{O}$, C, D for (complex) concepts, $?x, ?y, ?z$ for variables, c_{t_1}, c_{t_2} for concept terms, r_{t_1}, r_{t_2} for role terms, t, t' for individual terms, $q = \{\mathsf{at}_1, \ldots, \mathsf{at}_n\}$ for a query with n axiom templates over the query signature $\mathcal{S}_q = (N_C, N_R, N_I, V_C, V_R, V_I)$, $\Gamma_q^{\mathcal{O}}$ for the compatible mappings and $\Omega_q^{\mathcal{O}}$ or $\mathsf{ans}(\mathcal{O}, q)$ for the solution mappings of q over $\mathcal{O}$. Moreover, we use the term *conjunctive queries* to refer to standard conjunctive queries which allow for non-distinguished variables and the term *conjunctive instance queries* to refer to conjunctive queries without non-distinguished variables.

Chapter 3

Query Answering and Optimization Approaches

In this chapter, we present state-of-the-art query answering and optimization techniques as implemented in databases (Section 3.1) and knowledge bases (Section 3.2). The description of database query answering and optimization techniques in Section 3.1 aims at providing the intuition for the query ordering techniques described in the next chapter. The description of knowledge base query answering techniques and optimizations aims at describing alternative widely used techniques to the techniques developed in this thesis and places our work in context. In the next chapter (Section 4.6), we explain how the developed query ordering techniques can be employed by each of the approaches presented in this chapter.

3.1 Query Answering over Databases and Triple Stores

3.1.1 Databases

The procedure that is followed by a database management system to answer queries consists of the following steps:

1. The query given by the user in a database query language like SQL is first translated to a more useful representation so that it can be processed efficiently. One such representation is relational algebra and the translation is done by the query parser of the system.

2. Afterwards, the query optimizer searches several alternative equivalent execution plans and strategies for the query and chooses the one that can be evaluated most efficiently.

3. In the end the execution engine executes the query chosen by the query optimizer.

In the following, we briefly describe the query optimization module of database systems. In databases, query optimization [68, 2, 67, 20, 29] happens at several levels. At the logical (relational algebra) level, the system tries to find an expression (execution plan) that is equivalent with the given expression but which can be

executed more efficiently. For example, at this level the system may move a selection operator for a join relation in front of a base relation or it may order joins differently. Let $\sigma_{A=a}$ be a selection for attribute A, r, s relations and let us assume that s contains the attribute A whereas r does not. In this case we can use the equivalent expression $r \bowtie \sigma_{A=a}(s)$ instead of $\sigma_{A=a}(r \bowtie s)$, which generally leads to smaller intermediate relations as it reduces the tuples of the s relation before performing the join. At the physical level, the system tries to choose a detailed processing strategy for the query, such as the choice of an algorithm that will be used for the execution of an operator, the choice of the specific index that will be used. For example, at this level the system decides what kind of (physical) join operators like nested loop, sort-merge or hash join will be used based on the presence/absence of indexes, like B^+-trees or hash indexes and how the execution of the operators will be tuned.

The difference between the running time of a good strategy and that of a bad one can be huge [121]. In order to distinguish between different execution plans, the optimizer assigns a cost to each plan. This cost can be estimated in terms of different parameters, like disc access (how many block transfers to and from secondary memory are expected to be made) or CPU time. Disc access that is slow in comparison to memory access is usually prominent in the cost of processing a query. In order to estimate this cost accurately, query optimizers use statistical information for the database relations, such as relation sizes (number of tuples in relations), tuple sizes of relations in bytes, number of blocks in disc that contain tuples of a relation or the number of tuples of a relation that fit in a block. There are different ways one can estimate this cost; the simplest is by estimating the size of intermediate results. In reality, a cost estimation algorithm should take into account the actual cost of the algorithms used for the joins (different algorithms will incur different costs), whether one relation has an index which influences the cost or not, but this simple cost estimation is reasonable and accurate in many cases since the cost is approximately proportional to the number of I/Os which is generally monotonic in the number of retrieved tuples. An efficient query evaluation method should, therefore, aim at minimizing the tuples retrieved at each step.

We now focus on the problem of estimating an efficient join order (i.e., an order for the atoms of a query which yields the least cost). In order to estimate the cost of a join order, the database management system holds a set of statistics, like the cardinality of the relations, the number of distinct values for each attribute of the relations and, as we stated above, the system often aims at choosing plans that yield the least intermediate result sizes.

We now show how the size of a join between two relations can be estimated. Based on this, the cost of a query plan, i.e. a sequence of joins, can be easily computed by taking the sum of the estimated sizes of the intermediate joins. Let us assume that we have the relations $r(A, B)$ and $s(B, C)$, where A, B and C are attributes and we want to estimate the size of the join $r \bowtie s$. Let us assume that n_r denotes the size of r (number of tuples in r) and $V(A, r)$ the number of distinct values for the attribute A of r. In the general case (i.e., B is not a key for r and s), a tuple t of r is expected to produce $n_s/V(B, s)$ tuples in $r \bowtie s$, since this is the average number of tuples in s with a specific value for the B attribute. Hence, all tuples in r will produce $n_r * n_s/V(B, s)$ tuples in $r \bowtie s$. If we change the roles of r and s in the previous estimate we will get $n_s * n_r/V(B, r)$ tuples in $r \bowtie s$. If $V(B, r) \neq V(B, s)$ then the tuples in the difference of the two sets of estimates

are possibly invalid tuples that are not in the join. Hence the smallest of the two estimates is probably the most accurate.

In order to more accurately estimate relation and join sizes more complex techniques can be used such as histograms, parametric or sampling methods.

Methods based on histograms [89, 66] estimate join sizes based on detailed stored information about the underlying relations. A histogram can represent the number (or fraction) of the tuples having each value for an attribute of a relation. If there are many values for an attribute, then only the most frequent values may be recorded individually; the other values can be counted in groups. Several kinds of histograms have been proposed, the most common of which are the equi-width histograms, in which the range of values of an attribute is split into intervals or "buckets" of equal widths and the height of the histogram at an interval represents the number of values in the corresponding range or the frequency of each attribute value in the data and the equi-height histograms, in which the frequencies of the attribute values associated with each range is the same independent of the number of these attribute values in the range. Storing and maintaining detailed statistics about relations could be prohibitively costly in terms of space and computational overhead, particularly in the presence of updates.

Parametric methods [89] provide size estimates using statistical data distribution functions. In most cases these approaches place certain assumptions on the distribution of data, like the uniform data distribution assumption, and assume that data under different attributes/relations are independent. However, realistic data distribution could be rather arbitrary and complex and therefore cannot easily be simulated by pure mathematical models such as Poisson or Normal distribution. Hence parametric methods are rather limited in their use. A clear advantage of these methods is their efficiency as they require no disk I/Os and little computational and space overheads.

Sampling methods [100] collect information for size estimation by examining a small fraction of the database instances that are relevant for the query at run time. Sampling methods have been shown to produce rather accurate estimates regardless of the underlying data distributions and dependencies. Unlike parametric methods they do not require a priori assumptions about how the data fit in a probability distribution. Unlike histograms they do not require storing and maintaining detailed statistics about the data. They are robust in the presence of correlation of attributes, which generally allows accurate estimation for queries that involve many operators. Despite all these advantages, sampling is an expensive method which requires accessing tuples and relations several times at run time. As with similar such techniques there is a trade off between achieving good estimation accuracy (achieved by taking more samples) and high efficiency (achieved by taking less samples).

In order to find the best execution plan w.r.t. running time we need algorithms that construct plans over the execution plan space whose cost is determined by estimation techniques as discussed above. The enumeration of several alternative plans can be done by exhaustive search possibly enhanced by pruning techniques that exclude unlikely candidates for good solutions. Most state of the art methods employ dynamic programming techniques for enumerating plans in an appropriate order. According to dynamic programming, for n relations we find the cost for each subset containing $1, \ldots, n$ relations inductively, i.e., to compute the costs for relations of size $k+1$, $0 \leq k < n$, we take the ordering that leads to the least cost

for relations of size k and then add one more relation to the already constructed join order such that the total join expression yields the least cost. For relations of size n the join order that yields the least computed cost (more efficient join order) gives us the best way to compute the join of all the relations as well as the estimated cost of that method. Minimizing the total cost of the execution plan according to the underlying cost model is feasible but practical only for a limited number of joins.

Since enumerating all possible plans using dynamic programming is not practical, heuristic techniques can be used for reducing the number of plans for which their cost needs to be estimated but considering the least cost plan. For example, for finding a good join ordering, a greedy heuristic technique can be used, where we start by joining the pair of relations whose result has the smallest estimated cost and then repeat the process for the result of that join and the other relations in the (remaining) set to be joined. In a similar way, the selectivity of the join relation computed as the size of the join result to the size of the cartesian product of the two relations can be used. According to the minimum selectivity heuristic [129], at each step the relation that yields the minimum join selectivity when joined with the current join is added to the join computed so far. This heuristic is based on the assumption that good solutions are generally characterized by intermediate results with small cardinality. Apart from these, other heuristically based algorithmic enumeration techniques have been presented in the literature including branch and bound, hill climbing, simulated annealing, genetic programming [129, 29]. The branch and bound approach, for example, begins by using heuristics to find a good plan and then prunes plans for subqueries (sub-joins) which have a cost greater than the cost of the already found good plan, while updating the cost and best plan in case a better plan is found. Often these cheaper heuristic techniques produce very good execution plans but there is a risk of returning very poor plans.

In database literature static optimization-ordering techniques are mainly used (i.e., the query is ordered in the beginning before query execution). Dynamic ordering has also been proposed in the database literature in the context of adaptive query processing systems [41]. Dynamic ordering aims at overcoming the limitations of static optimization such as the lack of necessary statistics and of good selectivity estimates at compile time and the absence of knowledge for the runtime mappings of a query's variables at compile time. These techniques take into account changes that happen to the evaluation environment at runtime and modify the execution plan at runtime (i.e., they change the used operators for joins or the order in which the (remaining) query atoms are evaluated).

3.1.2 Triple Stores

Triple store query answering systems have a similar structure to database systems. A query parser parses the query to an abstract query, which is subsequently optimized. The optimization module searches for an efficient execution plan between several equivalent plans, which is afterwards executed by the query execution module. Query ordering and optimization techniques have recently been applied in the context of triple stores for queries that are posed over RDF graphs and expressed in the SPARQL query language. As in databases, query ordering techniques over RDF graphs are based on the estimation of the sizes of triple patterns and joins of triple patterns (i.e., the number of RDF triples that match a triple pattern or a join

of triple patterns) and aim at reducing the cardinality of intermediate results.

The work of Stocker et al. [131] is among the first works presenting a framework for selectivity based optimization of BGPs of SPARQL queries. The authors propose an algorithm that is a slight variation of the minimum selectivity algorithm described above and uses several kinds of heuristics together with precomputed statistics for the estimation of the sizes and selectivities of triple patterns and joined triple patterns. The selectivity of a triple pattern t is estimated by the formula $sel(t) = sel(s) \cdot sel(p) \cdot sel(o)$, where $sel(s)$, $sel(p)$, $sel(o)$ denote the selectivity of the subject, predicate and object respectively of the triple pattern, and which assumes that $sel(s)$, $sel(p)$ and $sel(o)$ are statistically independent. The authors estimate the sizes (and selectivities) of (bound or unbound) subjects and objects of triples patterns using statistics relating to the number of triples, the number of distinct subjects and histograms representing object values distributions for each predicate. Regarding the sizes of predicates they accurately precompute them. The sizes of joined triple patterns are precomputed by executing SPARQL queries for every pair of predicates that is related through a term of the RDF(S) schema vocabulary, in case such a vocabulary exists, and these sizes are cached for future use. The size estimates are accurate for triple patterns with unbound subjects and objects and they are used together with the size of restricted subjects or objects to estimate the size of patterns with bound subjects or objects.

The authors of RDF-3X [99] use a dynamic programming algorithm for determining an efficient join order which is based on two kinds of statistics for selectivity estimation. They use specialized histograms for RDF data, which can handle triple patterns and joins. The disadvantage is that these histograms are mostly based on the assumption that predicates are independent, which most of the time is not true. For example, the triple patterns

$$?x \text{ :studentOf } ?y.$$
$$?x \text{ :takesCourse } ?z.$$

are highly correlated since most objects that are students take a course. In other words, searching just one triple pattern is nearly as selective as searching for all of them. That is why they also use precomputed frequent join paths in the RDF graphs, which cover some cases of correlations between predicates but are time and space inefficient. When these frequent join paths are available they are used during query optimization, otherwise the specialized histograms are used assuming that predicates are not correlated.

In order to capture correlations between join predicates, which very commonly exist in RDF graph patterns, Neumann et al. [98] propose a novel statistical synopsis technique called characteristic sets. Characteristic sets are proposed as a more suitable technique than histograms for the estimation of the cardinality of (joined) triple patterns, which takes into account the heterogeneous and string oriented nature of RDF. In principle, characteristic sets capture the co-occurence of predicates with the same entities (subject or object) in the RDF data. Entities that have the same characteristic set tend to be semantically similar and this is the fact on which the authors are based to estimate the selectivity of join predicates. In [63] the authors also give emphasis to the dependencies (correlations) between predicates that exist in SPARQL basic graph patterns and propose the use of Bayesian networks and chain histograms to precompute statistics for star and chain RDF

paths which are subsequently used for the estimation of the cardinality of star and chain shaped graph patterns, which are common shapes for SPARQL basic graph patterns. Star graph patterns have the form of a number of triple patterns with different predicates sharing the same subject variable, while chain patterns are sequences of triple patterns where the object of a triple pattern is the subject of the next triple pattern. An example of a star pattern is the one given above, i.e., $?x$:studentOf $?y$, $?x$:takesCourse $?z$. An example of a chain pattern is the following: $?x$:takesCourse $?z$, $?z$:isTaughtBy $?y$, $?y$:teachesAt $?w$. For the estimation of the cardinality of arbitrarily shaped basic graph patterns the authors decompose them to a set of star and chain basic graph patterns and combine the precomputed statistics for star and chain paths to estimate the overall cardinality of the composite basic graph pattern. Kaoudi et al. [70] adapt the formula for the estimation of the size of a join that is used in databases and that has been described in Section 3.1 to work with triple pattern joins. In this work, the adapted formula for join size estimation, which is based on the number of triples that match the two triple patterns, and the number of distinct values that join variables of the triple patterns can take, is used for cost estimation in a distributed environment. Moreover, it is worth noting that in this work Kaoudi et al. propose a dynamic ordering algorithm that uses a variation of the minimum selectivity heuristic and seeks to construct query plans that minimize the number of intermediate results during query evaluation.

3.2 Query Answering over Ontologies

Query answering over ontologies differs from query answering over databases. In databases, it is assumed that we have complete knowledge about the instances of schema predicates (concept and role names), i.e., the closed world assumption is adopted. In Description Logics or generally First Order Logic ontologies, on the other hand, we adopt the open world assumption, i.e., it is assumed that we have an incomplete description of the domain; absence of information does not mean negative information.

A consequence of the fact that DL ontologies employ the open world assumption is that an ontology usually has more than one model and query answering over an ontology requires checking whether the (Boolean) query (created after the use of a mapping function) is satisfied in all these models of the ontology. This is in contrast to databases, in which the database instance created by the database schema and data defines exactly one model and in which query answering requires the check of this single model. This explains why query answering with respect to a DL ontology is a (computationally) harder task than query answering with respect to a database.

We now present an example which shows the effect that the open (ontologies) and closed (databases) world assumptions have on query answering caused by one (in databases) versus many models (in knowledge bases). Let us assume that we have an ontology $\mathcal{O}$ with $\mathcal{O} = \langle \emptyset, \{A(a), B(b)\} \rangle$. In this ontology it holds $\mathcal{O} \not\models A(b)$ and $\mathcal{O} \not\models \neg A(b)$; some models of $\mathcal{O}$ satisfy the assertion $A(b)$, i.e., if $\mathcal{I}$ is such a model $\mathcal{I} \models \mathcal{O}$ and $\mathcal{I} \models A(b)$, while some other models of $\mathcal{O}$ do not satisfy the assertion $A(b)$, i.e., if $\mathcal{I}$ is such a model $\mathcal{I} \models \mathcal{O}$ and $\mathcal{I} \not\models A(b)$, i.e., $\mathcal{I} \models \neg A(b)$. In contrast, the database instance $D = \{A(a), B(b)\}$ defines exactly one model, namely, $\mathcal{I}$, with $\Delta^{\mathcal{I}} = \{a, b\}$, $A^{\mathcal{I}} = \{a\}$, $B^{\mathcal{I}} = \{b\}$ and it holds $\mathcal{I} \models \neg A(b)$. The above means that a query $q = \{\neg A(?x)\}$ has no answer over $\mathcal{O}$, whereas it has the answer $?x \mapsto b$

over D.

In Description Logic ontology systems we usually do not adopt the Unique Name Assumption, which is adopted in database systems and according to which different individual names are assumed to be mapped to distinct elements in models.

Since the query ordering techniques usually depend on the employed reasoning algorithm and optimizations, we first briefly describe some well known optimization techniques for answering conjunctive (instance) queries. Each of these approaches can easily be adapted to work using our SPARQL parsing framework. For the techniques that are not applicable to OWL 2 DL ontologies, we briefly state the reasons why this is the case.

3.2.1 Materialization Techniques

Materialization techniques, also known as saturation techniques, have been applied for ontology based query answering. These techniques have been widely used in the context of RDFS for queries issued over RDF graphs. They work by extending the queried graph with all the relevant consequences that are implied under the RDFS semantics. This allows for using SPARQL's standard subgraph matching over the extended graph in order to compute answers under the RDFS semantics. An obvious disadvantage of materializing all consequences is that the extended graph can be quadratically larger than the original graph [133].

Saturation techniques can also be used for profiles of OWL. In particular, the OWL 2 RL [92] profile was designed such that rule-based implementations can apply a set of inference rules to make OWL RL consequences explicit. The queried ontology is first saturated by applying some completion rules and then the query is answered based on the produced assertion axioms [86]. We illustrate this by the following example.

Example 4. *Let us assume that we have an ontology with the following TBox $\mathcal{T}$ and ABox $\mathcal{A}$:*

$\mathcal{T} = \{$

$$\mathsf{Person} \sqcap \exists \mathsf{isHeadOf}.\mathsf{Department} \sqsubseteq \mathsf{Chair}, \tag{3.1}$$

$$\mathsf{Man} \sqcup \mathsf{Woman} \sqsubseteq \mathsf{Person}, \tag{3.2}$$

$$\mathsf{Chair} \sqsubseteq \mathsf{UniversityStaff} \tag{3.3}$$

$\}$

$\mathcal{A} = \{$

$$\mathsf{Woman}(mary), \tag{3.4}$$

$$\mathsf{isHeadOf}(mary, mathsDepartment) \tag{3.5}$$

$$\mathsf{Department}(mathsDepartment) \tag{3.6}$$

$\}$

and we want to answer the query

$$q = \{\mathsf{UniversityStaff}(?x)\}$$

We first compute the saturation of $\mathcal{A}$ w.r.t. to $\mathcal{T}$ by producing all implicit consequences, i.e., all entailed assertions. In particular, from Axioms (3.2) *and* (3.4) *we get* $\mathsf{Person}(mary)$ *and from this axiom and Axioms* (3.1), (3.5) *and* (3.6) *we get* $\mathsf{Chair}(mary)$. *Finally, from this axiom and Axiom* (3.3) *we get* $\mathsf{UniversityStaff}(mary)$. *Hence the saturation set $\mathcal{A}'$ is the following:*

$$\mathcal{A}' = \{\mathsf{Woman}(mary), \mathsf{isHeadOf}(mary, mathsDepartment), \mathsf{Person}(mary),\\ \mathsf{Department}(mathsDepartment), \mathsf{Chair}(mary), \mathsf{UniversityStaff}(mary))\}$$

Using $\mathcal{A}'$ we can easily see that the mapping $?x \mapsto mary$ is the only answer for q.

There are several well known RDF repository systems that perform some amount of OWL RL reasoning like AllegroGraph,[1] Apache Jena,[2] Virtuoso,[3] Oracle 11g,[4] OWLim,[5] Sesame.[6] The use of materialization techniques is problematic in case the data change frequently because in such case the frequent recomputation of the entailed implicit facts is required which is a costly operation.

Extending saturation approaches to OWL 2 DL necessarily leads to unsoundness and/or incompleteness of query answering since the presence of disjunctive information in OWL DL no longer allows for a single canonical extension of the queried ABox w.r.t. the TBox, i.e., there is no unique partial closure of the set of ABox assertions that contains all relevant OWL DL consequences. For example, the axioms $\mathsf{Person} \sqsubseteq \mathsf{Man} \sqcup \mathsf{Woman}$ and $\mathsf{Person}(mary)$ of an ontology cannot be satisfied in a unique canonical model that could be used to answer queries since in one model we would have that $mary$ is a man, whereas in another model, we would have that $mary$ is a woman. Note that if we restrict the saturation to facts that hold in all models, we are able to find sound and complete answers to conjunctive (instance) queries using this saturation set. However, this is not the case for queries that go beyond conjunctive queries. For example, let us assume that we have an ontology $\mathcal{O}$ with the following TBox and ABox:

$$\mathcal{T} = \{\mathsf{Student} \sqsubseteq \mathsf{BScStudent} \sqcup \mathsf{MScStudent} \sqcup \mathsf{PhDStudent}\}$$
$$\mathcal{A} = \{\mathsf{Student}(maria), \neg\mathsf{PhDStudent}(maria)\}$$

For the saturation $\mathcal{A}'$ of $\mathcal{O}$ it holds $\mathcal{A}' = \mathcal{A}$. Now, let us assume that we have the query

$$q = \{\mathsf{BScStudent}(?x) \sqcup \mathsf{MScStudent}(?x)\}$$

The answer to this query should be the mapping $?x \mapsto mary$, since $mary$ is a student but not a PhD student, which means that $mary$ is either a $\mathsf{BScStudent}$ or a $\mathsf{MScStudent}$ as a consequence of the TBox axiom. However, based on the saturated set there is no answer for this query. Moreover, the presence of existential quantification (on the right hand side of axioms) leads to the addition of assertions about new individuals in the saturated set and it cannot be guaranteed that the

[1] http://www.franz.com/agraph/allegrograph/
[2] http://jena.apache.org
[3] http://virtuoso.openlinksw.com
[4] http://www.oracle.com
[5] http://www.ontotext.com/owlim
[6] http://www.openrdf.org

saturated set will be finite. For example, if we have an ontology with TBox $\mathcal{T} = \{\mathsf{Person} \sqsubseteq \exists \mathsf{hasFather}.\mathsf{Person}\}$ and ABox $\mathcal{A} = \{\mathsf{Person}(mary)\}$ the saturated set would contain an infinite chain of hasFather relations between newly introduced individuals.

It is worth noting that for the OWL EL profile, there is no set of materialization rules given in the specification, but it recently has been shown that such techniques are nevertheless also applicable [85]. In this work, the saturation of the OWL EL ontology leads to the creation of a (saturated) datalog program that can be used to answer queries.

3.2.2 Query Rewriting Techniques

The OWL 2 QL profile was designed to allow query answering via *query rewriting*: a query over an OWL 2 QL TBox and a set of instance data stored in a data repository can be answered by rewriting the query w.r.t. the TBox and then answering the rewritten query in the data repository. OWL 2 QL, as discussed in Section 2.1.2 is based on DL-Lite$_R$, one of a family of Description Logics developed by Calvanese et al [17, 18]. Query rewriting techniques for DL-Lite are based on the notion of first order rewritability according to which, given a DL-Lite TBox $\mathcal{T}$ and a conjunctive query q, one can compute a First Order Logic query $q^{\mathcal{T}}$ such that, for every ABox $\mathcal{A}$, $\mathsf{ans}(\langle \mathcal{T}, \mathcal{A} \rangle, q) = \mathsf{ans}(\langle \emptyset, \mathcal{A} \rangle, q^{\mathcal{T}})$.

Query rewriting techniques focus on modifying the query rather than the queried data and are sound and complete for conjunctive queries. In order to apply the technique, the ontology is split into a schema or TBox and a data part or ABox. The data part can then be stored in a standard database, while the schema part is used to rewrite a query into a union of one or more conjunctive queries such that the resulting rewritten query can be evaluated over the data alone without taking the schema into account. Thus, standard database storage techniques and optimizations can immediately be used for query evaluation. Query rewriting techniques are widely used in the context of ontology based data access. In this context the user formulates a conjunctive query q that uses terms from the vocabulary of a TBox $\mathcal{T}$ and the ontology based data access system rewrites q using $\mathcal{T}$ to a new query $q^{\mathcal{T}}$ such that for any ABox $\mathcal{A}$ the answers to q over the consistent $\langle \mathcal{T}, \mathcal{A} \rangle$ are the same as the answers of $q^{\mathcal{T}}$ over $\mathcal{A}$.

Calvanese et al. [18] proposed and implemented in the QuOnto system[7] [3] the first algorithm for query rewriting for the family of DL-Lite description logics which uses the TBox axioms as rewriting rules and computes a union of conjunctive queries that is a rewriting of q w.r.t. $\mathcal{T}$. Intuitively, the algorithm uses the axioms in $\mathcal{T}$ to replace concepts and roles in the query by concepts and roles that imply them.

Example 5. *If we assume that we have the following TBox and ABox:*

$$\mathcal{T} = \{\mathsf{Student} \sqsubseteq \mathsf{UniversityMember}, \mathsf{takesGraduateCourse} \sqsubseteq \mathsf{takesCourse}\}$$
$$\mathcal{A} = \{\mathsf{Student}(mary), \mathsf{takesGraduateCourse}(mary, maths)\}$$

and that we ask the conjunctive instance query

$$q = \{\mathsf{takesCourse}(?x, ?y), \mathsf{UniversityMember}(?x)\}$$

[7] http://www.dis.uniroma1.it/quonto/

the proposed algorithm applies the TBox axioms backwards producing new subqueries that can produce answers when evaluated over databases. In particular, we can apply $\mathsf{takesGraduateCourse} \sqsubseteq \mathsf{takesCourse}$ *to the* $\mathsf{takesCourse}(?x, ?y)$ *query atom and obtain a new query*

$$q' = \{\mathsf{takesGraduateCourse}(?x, ?y), \mathsf{UniversityMember}(?x)\}$$

By applying now $\mathsf{Student} \sqsubseteq \mathsf{UniversityMember}$ *to* q' *we get the query*

$$q'' = \{\mathsf{takesGraduateCourse}(?x, ?y), \mathsf{Student}(?x)\}$$

In the same way by applying $\mathsf{Student} \sqsubseteq \mathsf{UniversityMember}$ *to* q *we get*

$$q''' = \{\mathsf{takesCourse}(?x, ?y), \mathsf{Student}(?x)\}$$

After applying to each query atom all the applicable axioms, we can obtain a union of conjunctive queries that retrieves all the query answers when posed over the ABox data only. In our example, the union of conjunctive queries that the algorithm returns contains the four queries q, q' q'' *and* q'''. *The answer over the ABox* $\mathcal{A}$ *given above is* $?x \mapsto mary$, $?y \mapsto maths$.

The data complexity of query answering w.r.t. DL-Lite is in AC_0 [18, 104], which means that query evaluation can be performed efficiently. However, the size of the rewritten queries can be large making in some cases the performance not satisfactory in practice. In particular, the size of the rewritings is in the order of $(|\mathcal{T}| \cdot |q|)^q$, where $|\mathcal{T}|$ and $|q|$ are the sizes of the TBox and the original conjunctive query, respectively. In an effort to reduce the huge number of produced rewritings, an alternative resolution based query rewriting technique was proposed by Pérez-Urbina et al. [107] and was implemented in the Requiem system.[8] This approach rewrites the initial conjunctive query to a union of conjunctive queries which is, generally, smaller in size than the rewriting produced by QuOnto as it is shown by a thorough experimental evaluation [107]. Several other optimization techniques have been developed for reducing the size of the rewritten query even further [39, 21, 112]. Chortaras et al. [21] implemented the Rapid system, which applies the rewriting rules only in those cases that lead to the creation of useful, non-redundant conjunctive queries avoiding many query subsumption tests. However, even when using optimization techniques that try to reduce the number of the rewritings, the rewritings' size are still worst case exponential in the size of the original query and hence cannot always be evaluated efficiently using existing database technology.

Rosati et al. [114] proposed a quite different query rewriting technique for DL-Lite, which generates a non-recursive datalog program instead of a union of conjunctive queries and was implemented in the Presto system. This allows the "hiding" of the exponential blow-up inside the rules instead of generating explicitly the disjunctive normal form delegating, in this way, part of the computational complexity of this task to the (deductive) database system. However, the technique is still worst case exponential. Gottlob et al. [40] presented an approach which produces a polynomial time rewriting but it uses additional entities.

It should be noted that query rewriting techniques can also be used with the OWL EL profile, but the rewriting then produces a datalog query instead of a union

[8] `http://www.cs.ox.ac.uk/isg/tools/Requiem/`

of conjunctive queries [113, 108]. Deductive databases can then be used to evaluate the datalog query over the data.

Finally, an approach, called *combined rewriting* has been proposed by Kontchakov et al. [82] for the description logic DL-Lite and by Lutz et al. [87] for the description logic EL. Their approach is quite different from the standard query rewriting approach since the proposed algorithm rewrites both the query and the ABox w.r.t. the TBox. Kontchakov et al. [82] propose a polynomial rewriting over the data expanded by applying DL-Lite axioms (without role inclusions) from the queried ontology and introducing a small number of fresh individuals. This combined approach is not efficient in the presence of updates since in such cases the expensive rewriting (extension) of the data w.r.t. the TBox would need to be done frequently.

Query rewriting techniques have also been developed and implemented in a system called Clipper for more expressive Horn DLs, like Horn-$\mathcal{SHIQ}$ [27].

Current query rewriting techniques cannot be applied to more expressive languages such as OWL 2 DL because these techniques are based on the canonical model property of the considered DL, according to which, from a satisfiable ontology $\mathcal{O}$, one can construct a canonical model $\mathcal{I}_{\mathcal{O}}$ such that $\mathsf{ans}(\mathcal{I}_{\mathcal{O}}, q) = \mathsf{ans}(\mathcal{O}, q)$ for every positive First Order Logic query q. In other words, in DLs that enjoy the canonical model property, we do not need to look at all models to answer a conjunctive query; the canonical model suffices and can be used to find all query answers to conjunctive queries. The presence of disjunctive information results in the existence of non-homomorphically related models that differ in the queries they entail and hence cannot be represented by a unique canonical model. For example, as it has been stated in Section 3.2.1 the following two axioms of an ontology $\mathcal{O}$

$$\mathsf{Person} \sqsubseteq \mathsf{Man} \sqcup \mathsf{Woman}$$
$$\mathsf{Person}(mary)$$

cannot be satisfied in a unique canonical model that could be used to answer queries since in one model we would have that *mary* is a man, whereas in another model, we would have that *mary* is a woman.

3.2.3 Techniques for Expressive Ontological Knowledge

For more expressive, (possibly) non-deterministic ontologies query answering is mainly based on model building procedures such as tableau and hypertableau calculi. There are several widely used reasoners that use (hyper)tableau techniques and optimizations to answer queries, some of which are Pellet[9] and Racer.[10] A couple of optimizations for conjunctive instance queries over OWL DL ontologies have been presented by Sirin et al. [124]. These take advantage of instance retrieval optimization techniques that have been developed for tableau reasoners. Query answering optimization techniques over OWL 2 DL ontologies using (hyper)tableau reasoners are also the focus of this thesis. These techniques have been implemented using the HermiT reasoner.[11] In Section 4.4, we compare the query ordering techniques developed for Pellet [124] and Racer [47] with those developed in the current thesis.

[9] `http://clarkparsia.com/pellet/`
[10] `http://www.racer-systems.com`
[11] `http://hermit-reasoner.com`

In order to better understand the differences and given the fact that query ordering techniques depend to some extent on the employed reasoning algorithm and optimizations, in the following, we briefly describe some well known optimization techniques for instance retrieval employed by Pellet and Racer. Since consistency checking is an expensive operation and query answering is reduced to consistency checking, the goal of these optimizations is to reduce the number of performed consistency checks by cheaply finding obvious (non-)instances of query atom concepts and roles.

For finding obvious instances of concepts and roles usually a precompletion, i.e., a set of assertions that is obtained by the application of only deterministic tableau rules is exploited. Since a precompletion contains only the deterministic assertions or the assertions that have been created by deterministic rules, it is obvious that the precompletion represents a common part of every model of the queried ontology. This means that one can (cheaply) determine obvious instances of concepts and roles by looking at this precompletion set. This precompletion set can be extracted from the pre-model construction procedure during the initial ontology consistency check and it can be cached for future use. This check is usually performed in the beginning before query answering takes place; any logical statement is entailed by an inconsistent ontology. In order to be able to extract deterministically derived assertions from the pre-model, one needs to keep dependency information for each concept (role) assertion that appears in the pre-model [137, 7], i.e., one needs to keep information regarding the concept (role) assertions that caused the creation of other concept (role) assertions due to the application of tableau rules. If such information is cached for each concept (role) assertion, the cases where a non-deterministic choice is made can easily be identified and assertions which depend on non-deterministic choices can be excluded from the precompletion set. Optimized DL reasoners keep such statistics for optimized (dependency directed) backtracking while searching the non-deterministic branches during the consistency check.

For finding obvious non-instances the pseudo model merging technique [46] can be used, which is a (cheap) sound but incomplete technique to determine if an individual a is not an instance of a concept C without considering concept and role assertions for other individuals of the queried ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and avoiding many expensive consistency checks. According to this technique assuming that we have a pseudo model for the (satisfiable) concept $\neg C$ (which can be built after a concept satisfiability test) and a pseudo model for the individual a (which can be extracted from the label of the individual in the pre-model built for the initial consistency check of the ontology) we check if the pseudo model for $\neg C$ can be merged with the pseudo model of the individual a; if there are no concept and role interactions between the two pseudo models that lead to a clash, this means that the ABox $\mathcal{A} \cup \{a : \neg C\}$ is consistent w.r.t. $\mathcal{T}$ and hence we can conclude that a is not an instance of C.

Apart from caching pseudo models and exploiting precompletion or pre-model information, binary instance retrieval [48] is generally used to reduce the individuals that need to be tested in the case of a query atom of the form $C(?x)$. Binary instance retrieval uses a divide and conquer technique and tries to decide whether a set of individuals are non-instances of a concept using one consistency check. In particular, for testing for possible instances of a concept C, the candidate instance set is split into two partitions. For each partition, a single consistency check is performed. If

the consistency check is successful, it is safe to consider all individuals belonging to the partition as non-instances of the tested concept C. Otherwise, we further split the partition and process the resulting partitions in the same way. In this case, one performs one consistency check to potentially determine several (non-)instances of C. The method is highly efficient if the partitioning is done in a way that maximizes the chance that non-instances of the queried concept all belong to the same partition.

In the next chapter, we discuss the optimization techniques used for query answering in our context. Even in the presence of optimizations, techniques based on consistency checking over OWL 2 DL ontologies suffer from scalability issues, i.e., a query answering system that is based on the reduction of query answering to ABox consistency checking does not scale well, when the amount of data increases even when optimizations are employed.

In an effort to improve scalability, summarization and refinement techniques have been developed by Dolby et al. [24, 25] and implemented in the SHER system [26], which determine query answers testing a group of individuals in each tableau consistency check rather than a single individual. According to these techniques, a summary ABox is computed from the original ABox before the beginning of the query evaluation procedure. The summary ABox is created by aggregating individuals that belong to the same concepts into a single summary individual. To answer an instance query, every individual in the summary ABox is used to instantiate the query. For each such individual the negation of the instantiated query concepts are added to the summary ABox and the summary ABox is tested for consistency. If it is consistent then all individuals that are mapped to the summary individual are not answers to the query. In case of inconsistency, it holds that either a subset of the individuals that are mapped to the summary individual are answers to the query or the inconsistency was the result of the summarization procedure (this holds because individuals are aggregated based only on the concepts, i.e., role assertions are not taken into account when the summary ABox is created) in which case a refinement step follows in which the summary ABox becomes more precise based on inconsistency justifications (minimal assertion sets implying the inconsistency). To find answers to role atoms, Dolby et al. use a more complex procedure. A conjunctive instance query is answered by joining the answers of every query concept or role. Note that summarization and refinement techniques generally provide a more efficient partitioning of individuals than divide and conquer techniques such as binary instance retrieval described above.

In order to deal with large amounts of data and remain scalable, a different approach has been developed by Hustadt et al. [64] and Motik et al. [91]. In this work, the reduction of a $\mathcal{SHIQ}$ knowledge base to a disjunctive datalog program that entails the same set of ground facts as the original knowledge base and hence gives the same answers to conjunctive instance queries has been explored. Using the datalog program one can take advantage of optimization methods from deductive databases, such as the join order optimization or the magic set transformation [11] for finding answers to queries more efficiently. This work resulted in the creation of the KAON2 reasoner.[12]

[12] `http://kaon2.semanticweb.org`

3.2.4 Approximation Techniques

In practice, current systems that perform query answering over expressive Description Logic languages are not as scalable as needed, when they have to answer queries over large amounts of data. Hence, in order to achieve practicality, approximate query answering systems have been proposed, which are based on approximate reasoning algorithms. Approximate systems usually sacrifice soundness and/or completeness in order to speed-up the query answering performance; approximate query answering algorithms can either be (i) sound and incomplete, i.e., they under-approximate the set of certain or known query answers, or (ii) they can be complete but unsound, i.e., they over-approximate the set of certain answers or (iii) they can be unsound and incomplete. Typical examples of such algorithms rewrite the queried ontology into a simpler logic in such a way that computing the results over the simplified ontology yields the desired under- or over-approximation. For example, for the evaluation of a conjunctive query q over an ontology $\mathcal{O}$, an approximate query answering algorithm could evaluate q over an ontology $\mathcal{O}'$, where $\mathcal{O}'$ is created from $\mathcal{O}$ after the removal of a set of axioms. It is obvious that answering q over $\mathcal{O}'$ results in an under-aproximation of query answers of q over $\mathcal{O}$ (due to the monotonicity property of description logics), i.e., the query answering procedure is sound but incomplete. Several approximate query answering algorithms have been proposed in the literature.

Hitzler et al. [54] and Tserendorj et al. [138] describe an approximate system for query answering over $\mathcal{SHIQ}$ description logics called Screech. Screech uses KAON2 to transform a $\mathcal{SHIQ}$ ontology into a disjunctive datalog program, which gives the same answers to instance queries as the original ontology, and it then transforms the disjunctive datalog program to a non-disjunctive one by changing disjunctions to conjunctions. Afterwards, it uses this program for approximately answering queries; the used approximation algorithm is complete but not always sound for instance queries over $\mathcal{SHIQ}$ ontologies. Other variants of the algorithm remove disjunctive rules instead of converting disjunctions to conjunctions and, hence, are sound but incomplete or they hold only one of the disjuncts of disjunctive rules and throw away the rest, resulting in unsound and incomplete procedures for ABox reasoning.

Pan et al. [102] present an approximation algorithm for computing answers to conjunctive queries over OWL DL ontologies. They propose to approximate the queried OWL DL ontology $\mathcal{O}$ with a DL-Lite ontology $\mathcal{O}_L$ that consists of the (finite) set of DL-Lite axioms created using only terms from the vocabulary of $\mathcal{O}$ and which are entailed by $\mathcal{O}$. The authors call this set an entailment set. This DL-Lite ontology can be used by scalable rewriting techniques to find query answers as discussed in Section 3.2.2. In such a way sound (but generally incomplete) answers to conjunctive queries are produced. It should be stated though that in the case of conjunctive queries that do not contain non-distinguished variables, i.e., for conjunctive instance queries, the described method provides sound as well as complete answers. Pan et al. [103] use these entailment sets to additionally produce an over-approximation of query answers by removing atoms with non-distinguished variables from the query. Since such over-approximation can produce many unsound answers, they propose a technique, which is based on transforming the ontology, based on the query, apart from the query only, to create stricter over-approximations. For the creation of the entailment sets they use a fully-fledged OWL DL reasoner for checking entailment of DL-Lite axioms over OWL DL ontologies, which can be expensive.

Kaplunova et al. [71] approximate a $\mathcal{SHI}$ TBox $\mathcal{T}$ to an OWL 2 QL TBox $\mathcal{T}_L$ such that for any ABox $\mathcal{A}$ containing only assertions about concept names the answers to a concept instance query q over $\mathcal{T}$ is a subset of the answers of q over $\mathcal{T}_L$, i.e., the answers of q over $\mathcal{T}_L$ is an over-approximation of the answers over the queried $\mathcal{SHI}$ ontology. After some preprocessing, the authors transform non-deterministically each axiom $C \sqsubseteq D$ in $\mathcal{T}$ to a DL-Lite axiom $C' \sqsubseteq D'$ such that $C \sqsubseteq C'$ and $D' \sqsubseteq D$. The proposed procedure can lead to the creation of inconsistent ontologies from consistent ones after the approximation and, moreover, the non-determinism involved in the transformation of axioms to DL-Lite means that approximating $\mathcal{O}$ can be expensive.

ABox materialization systems such as OWLim, which, as discussed in Section 3.2.1, perform sound and complete query answering for conjunctive (instance) queries over OWL 2 RL ontologies, behave as approximate query answering systems when they have to answer queries over more expressive ontologies. For ontologies outside the OWL 2 RL profile, these systems are sound for conjunctive queries but they are generally incomplete as they disregard the ontology axioms that are not in the OWL 2 RL fragment. Some of these systems additionally handle certain kinds of axioms that fall outside the OWL 2 RL fragment. In several application domains, application developers that use such incomplete systems are interested in knowing "how incomplete" a system is, i.e., the degree of (in)completeness of a system w.r.t. a TBox of some expressivity and a query. This information is also useful when application developers want to compare the degree of (in)completeness of several approximate query answering systems and choose the one that better satisfies their needs taking into account the trade-off between scalability and completeness. Cuenca Grau et al. [43] address this issue by a data generation technique; the authors generate a collection of datasets such that if a system is complete for a given TBox $\mathcal{T}$ and query q and each of the generated datasets, then it is complete for $\mathcal{T}$, q and *any* dataset. These generated datasets can be used to provide completeness guarantees. However, providing such guarantees is not always possible. That is why Cuenca Grau et al. further define conditions on ontologies and queries under which systems behave as complete reasoners.

A recent approximate query answering technique for OWL 2 DL ontologies has been proposed by Zhou et al. [143] and it is based on the use of an OWL 2 RL reasoner to find an under- and an over-approximation of query answers. Given an ontology $\mathcal{O}$ and a conjunctive query q, the under-approximation of query answers is found by using the OWL 2 RL reasoner to answer q over $\mathcal{O}$, i.e., ontology axioms outside OWL 2 RL are disregarded. To compute the over-approximation, Zhou et al. approximates the queried OWL 2 DL ontology $\mathcal{O}$ with an OWL 2 RL ontology $\mathcal{O}_{RL}$ such that $\mathcal{O}_{RL} \models \mathcal{O}$. The RL ontology is created by a sequence of steps that are roughly described below. First, $\mathcal{O}$ is transformed to a disjunctive datalog program using a variant of structural transformation of First Order Logic [96]. Then the disjunctive datalog program is transformed to a non-disjunctive one by transforming disjunctions in the head of the rules to conjunctions and by using fresh individuals to skolemize existentially quantified variables. Finally, the datalog program is transformed back into an OWL 2 RL ontology by rolling up datalog rules to OWL 2 RL axioms and eliminating non-RL axioms. An RL reasoner is then used to find an over-approximation of query answers. The tuples in the over- minus the under-approximation set are possible query answers and for completeness we need

to test these using an OWL 2 DL reasoner. Since this is expensive for huge datasets, Zhou et al. [144] present a backward chaining technique applicable to Horn OWL 2 ontologies for identifying a subset of the ontology that is sufficient for determining whether these possible answers are real ones using an OWL 2 DL reasoner such as HermiT.

Since the query answering techniques that we use can be seen as query approximation techniques, in the next chapter, we explain how they differ from some of the above described approximate systems and how the developed, in this thesis, approach can be used with approximate systems.

Chapter 4

Query Answering Optimizations

4.1 Motivation

A straightforward algorithm to compute the answers for a query q is to test, for each mapping μ, whether $\mathcal{O} \models \mu(q)$. Since only terms that are used in $\mathcal{O}$ can occur in the range of a mapping μ for q over $\mathcal{O}$, there are finitely many mappings to test. In the worst case, however, the number of mappings that have to be tested is still exponential in the number of variables in the query. Such an algorithm is sound and complete if the reasoner used to decide entailment is sound and complete since we check all mappings for variables that can constitute actual solution mappings.

Optimizations cannot easily be integrated into the above sketched algorithm since it uses the reasoner to check for the entailment of the instantiated query as a whole and, hence, does not take advantage of relations or dependencies that may exist between the individual axiom templates in q. For a more optimized evaluation, one can evaluate the query axiom template by axiom template. Initially, the solution set contains only the identity mapping, which does not map any variable to a value. One then picks the first axiom template, extends the identity mapping to cover the variables of the chosen axiom template and then uses a reasoner to check which of the mappings instantiate the axiom template into an entailed axiom. One then picks the next axiom template and again extends the mappings from the previous round to cover all variables and checks which of those mappings lead to an entailed axiom. Thus, axiom templates which are very selective and are only satisfied by very few solutions reduce the number of intermediate solutions. Choosing a good execution order, therefore, can significantly affect the performance.

For example, let $q = \{A(?x), r(?x, ?y)\}$ with $?x, ?y \in V_I$. The query belongs to the class of conjunctive instance queries. We assume that the queried ontology contains 100 individuals, only 1 of which belongs to the concept A. This A instance has 1 r-successor, while we have overall 200 pairs of individuals related with the role r. If we first evaluate $A(?x)$, we test 100 mappings (since $?x$ is an individual variable), of which only 1 mapping satisfies the axiom template. We then evaluate $r(?x, ?y)$ by extending the mapping with all 100 possible mappings for $?y$. Again only 1 mapping yields a solution. For the reverse axiom template order, the first axiom template requires the test of $100 \cdot 100$ mappings. Out of those, 200 remain to be checked for the second axiom template and we perform $10,200$ tests instead of just 200. Note also that the number of intermediate results when the query is evaluated in the order $A(?x)$, $r(?x, ?y)$ is smaller than when it is evaluated in the

reverse order (2 versus 201).

As it has been discussed in Section 3.1 in the context of databases or triple stores, cost-based ordering techniques for finding an optimal or near optimal join ordering have been widely applied [129, 131]. These techniques involve the maintenance of a set of statistics about relations and indexes, e.g., number of pages in a relation, number of pages in an index, number of distinct values in a column, together with formulas for the estimation of the selectivity of predicates and the estimation of the CPU and I/O costs of query execution that depends amongst others, on the number of pages that have to be read from or written to secondary memory. The formulas for the estimation of selectivities of predicates (result output size of query atoms) estimate the data distributions using histograms, parametric or sampling methods or combinations of them. Ordering strategies as implemented in databases or triple stores are, however, not directly applicable in our setting. In the presence of expressive schema level axioms, we cannot rely on counting the number of occurrences of explicitly given triples, since the inferred triples should also be taken into account. We also cannot, in general, precompute all relevant inferences to base our statistics on materialized inferences since materialization techniques are not sound and complete procedures for answering (complex) queries over ontologies in expressive languages as discussed in Section 3.2.1. Furthermore, to order queries over ontologies we should not only aim at decreasing the number of intermediate results, but also take into account the cost of checking or computing the solutions. This cost can be very significant with OWL reasoning and its precise estimation before query evaluation is difficult as this cost takes values from a wide range, e.g., due to nondeterminism and the high worst-case complexity of the standard reasoning tasks. For example, as it has already been stated the description logic $\mathcal{SROIQ}$, which underpins the OWL 2 DL standard, has a worst case complexity of 2-NExpTime [72] and typical implementations are not worst case optimal. The hypertableau algorithm that we use has a worst case complexity of 3-NExpTime in the size of the ontology [72, 96]. For several kinds of axiom templates we can, however, directly retrieve the solutions from the reasoner instead of checking entailment. For example, for $C(?x)$, reasoners typically have a method to retrieve concept instances. Although this might internally trigger several tests, most methods of reasoners are highly optimized and avoid as many tests as possible. Furthermore, reasoners typically cache several results such as the computed concept hierarchy and retrieving sub-concepts can then be realized with a cache look-up. Thus, the actual execution cost might vary significantly. Notably, we do not have a straight correlation between the number of results for an axiom template and the actual cost of retrieving the solutions as is typically the case in triple stores or databases. This requires cost models that take into account the cost of the specific reasoning operations as well as the number of results.

As motivated above, we distinguish between *simple* and *complex* axiom templates. Simple axiom templates are those that correspond to dedicated reasoning tasks and are evaluated efficiently. The evaluation of complex axiom templates might require iterating over the compatible mappings and by checking entailment for each instantiated axiom template. An example of a complex axiom template is $(\exists r.?x)(?y)$.

Definition 8 (**Simple and Complex Axiom Templates**). *Let c_{t_1}, c_{t_2} be concept terms, r_{t_1}, r_{t_2} be role terms and t, t' be individual terms. The set of* simple axiom

Algorithm 1 evaluate$(\mathcal{O}, q)$

Input: $\mathcal{O}$: the queried $\mathcal{SROIQ}$ ontology
q: a query over $\mathcal{O}$
Output: a set of solutions for evaluating q over $\mathcal{O}$

```
1:  S := {μ0 | dom(μ0) = ∅}
2:  for i = 1, ..., n do
3:    R := ∅
4:    for each μ ∈ S do
5:      if isSimple(at_i) and Var(at_i) \ dom(μ) ≠ ∅ then
6:        R := R ∪ {μ' ∪ μ | μ' ∈ callSpecificReasonerTask(μ(at_i))}
7:      else if Var(at_i) \ dom(μ) = ∅ then
8:        if O ⊨ μ(at_i) then
9:          R := R ∪ {μ}
10:       end if
11:     else
12:       B := {μ' | μ'(?x) = a, for ?x ∈ Var(at_i) \ dom(μ),
                    a ∈ N_C^O or a ∈ N_R^O or a ∈ N_I^O and
                    μ'(?y) = μ(?y) for ?y ∈ dom(μ)}
13:       for each μ' ∈ B do
14:         if O ⊨ μ'(at_i) then
15:           R := R ∪ {μ}
16:         end if
17:       end for
18:     end if
19:   end for
20:   S := R
21: end for
22: return R
```

templates *contains templates of the form:* $c_{t_1} \sqsubseteq c_{t_2}$, $r_{t_1} \sqsubseteq r_{t_2}$, $c_{t_1}(t)$, $r_{t_1}(t, t')$, $t \approx t'$. *All the rest are* complex axiom templates. *The function* isSimple(at) *takes an axiom template* at *and returns* true *if it is simple; otherwise it returns* false.

Based on the above, Algorithm 1 is a simple general algorithm showing the proposed procedure to answer a query q using any (hyper)tableau reasoner. For a simple axiom template, which contains so far unbound variables, we call a specialized reasoner method to retrieve entailed results, i.e., mappings for unbound variables (callSpecificReasonerTask in line 6). We now give the general outline of the algorithm; in the next section we explain in more detail how the method callSpecificReasonerTask works. For templates with all their variables bound, we check whether the mappings lead to entailed axioms (lines 11 to 18). Otherwise, for complex templates that do not have all their variables bound, we check which of the compatible mappings lead to entailed axioms.

4.2 Preprocessing for Information Extraction

In this section, we first present a way of preprocessing the queried ontology to extract information (Section 4.2.1) that is useful for ordering the axiom templates

in a query (Section 4.3). We afterwards propose a clustering of individuals based on the information extracted from the queried ontology (Section 4.2.1.1). This preprocessing is useful for axiom templates of the form $c_{t_1}(t)$, $r_{t_1}(t, t')$, or $t \approx t'$, where c_{t_1} is a concept term, r_{t_1} is a role term and t, t' are individual terms.

4.2.1 Information Extraction from Reasoner Models

The first step in the ordering of query atoms is the extraction of statistics by exploiting information generated by reasoners. We use the labels of an initial pre-model to provide information about the concepts the individuals belong to or the roles with which one individual is connected to another one. We exploit this information similarly as was suggested for determining known or possible (non-)subsumers for concepts during classification [33].

We first use an example to show how a hypertableau algorithm creates a pre-model of a satisfiable ontology and how information in this pre-model is exploited for creating ordering statistics. Let us assume that we have an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ as shown below:

$$\begin{aligned} \mathsf{Student} &\sqsubseteq \forall \mathsf{takesCourse}.\mathsf{Course} \\ \mathsf{Student} &\sqsubseteq \mathsf{GraduateStudent} \sqcup \mathsf{UndergraduateStudent} \\ \mathsf{GraduateStudent} &\sqsubseteq \forall \mathsf{takesCourse}.\mathsf{GraduateCourse} \\ &\mathsf{Student}(mary) \\ &\mathsf{takesCourse}(mary, maths) \end{aligned}$$

This ontology is translated to the following DL-clauses:

$$\mathsf{Student}(x) \wedge \mathsf{takesCourse}(x, y) \rightarrow \mathsf{Course}(y) \tag{4.1}$$
$$\mathsf{Student}(x) \rightarrow \mathsf{GraduateStudent}(x) \vee \mathsf{UndergraduateStudent}(x) \tag{4.2}$$
$$\mathsf{GraduateStudent}(x) \wedge \mathsf{takesCourse}(x, y) \rightarrow \mathsf{GraduateCourse}(y) \tag{4.3}$$
$$\rightarrow \mathsf{Student}(mary) \tag{4.4}$$
$$\rightarrow \mathsf{takesCourse}(mary, maths) \tag{4.5}$$

We start constructing a derivation using the derivation rules of the hypertableau calculus. The initial set of assertions is

$$\mathcal{A}_0 = \{\mathsf{Student}(mary), \mathsf{takesCourse}(mary, maths)\} \tag{4.6}$$

By applying clause (4.1) to $\mathcal{A}_0$ we get

$$\mathcal{A}_1 = \{\mathsf{Student}(mary), \mathsf{takesCourse}(mary, maths), \mathsf{Course}(maths)\} \tag{4.7}$$

By applying clause (4.2) to $\mathcal{A}_1$ we get

$$\begin{aligned} \mathcal{A}_2 = \{&\mathsf{Student}(mary), \mathsf{takesCourse}(mary, maths), \\ &\mathsf{Course}(maths), \mathsf{GraduateStudent}^1(mary)\} \end{aligned} \tag{4.8}$$

Note that the assertion $\mathsf{GraduateStudent}(mary)$ has a superscript. This superscript is used to denote that this assertion has been derived non-deterministically. The other alternative would be to add the assertion $\mathsf{UndregraduateStudent}(mary)$ to $\mathcal{A}_2$.

The hypertableau algorithm makes a random choice between the two assertions when both have concept names as predicates as is the case in the example above. In case we have a combination of concept name atoms and existential concept atoms, like $A(x) \vee \exists r.B(x)$ as the consequent atoms of a DL clause, the algorithm first adds the instantiated concept name assertions to the derivation set using the Hyp-rule of the hypertableau calculus and in case this leads to clashes it then tries the existential concept assertions. The intuition behind this is that the addition of concept names in the labels of individuals can lead to less complex models and hence it may be efficient to try the instantiated concept atoms first. By applying clause (4.3) to $\mathcal{A}_2$ we get

$$\begin{aligned}\mathcal{A}_3 = \{&\mathsf{Student}(mary), \mathsf{takesCourse}(mary, maths), \mathsf{Course}(maths),\\ &\mathsf{GraduateStudent}^1(mary), \mathsf{GraduateCourse}^1(maths)\}\end{aligned} \tag{4.9}$$

The assertion $\mathsf{GraduateCourse}(maths)$ is assigned the same superscript as the assertion $\mathsf{GraduateStudent}(mary)$ to denote that it was derived by a hypertableau rule using the non-deterministically derived assertion $\mathsf{GraduateStudent}(mary)$. In $\mathcal{A}_3$ there is no clash and no more rule is applicable; $\mathcal{A}_3$ is called a *pre-model*. In this pre-model there are assertions that are derived deterministically such as:

$$\mathsf{Student}(mary), \quad \mathsf{takesCourse}(mary, maths), \quad \mathsf{Course}(maths)$$

and assertions that are derived non-deterministically such as:

$$\mathsf{GraduateStudent}(mary),\ \mathsf{GraduateCourse}(maths).$$

We keep the known (K) and possible instances (P) for each concept and role. In our example, we have

$$K[\mathsf{Student}] = \{mary\}, \quad K[\mathsf{Course}] = \{maths\}, \quad K[\mathsf{takesCourse}] = \{\langle mary, maths\rangle\}$$

$$P[\mathsf{GraduateStudent}] = \{mary\}, \quad P[\mathsf{GraduateCourse}] = \{maths\}$$

Since it is not very obvious, we further show two examples of how a role can have possible instances. Let us assume that we have an ontology

$$\mathcal{O} = \{A \sqsubseteq \exists r.\{b\} \sqcup \exists r.\{c\}, A(a)\}$$

and that we want to check whether it is consistent. Let us assume that a pre-model is constructed, in which the concept assertion $\exists r.\{b\}(a)$ is non-deterministically chosen to be added. Using this pre-model we get that b is a possible r-successor of a. Let us now see a more complex example that shows how possible instances for roles are created using at most cardinality restrictions and role hierarchies. Let us assume that we have an ontology

$$\mathcal{O} = \{A \sqsubseteq \leq 2s.\top, r \sqsubseteq s, s(a, b_1), s(a, b_2), r(a, b_3), A(a)\}.$$

Since $\mathcal{O} \models r \sqsubseteq s$, the pre-model contains the assertion $s(a, b_3)$ and since we have three s-successors of a and a is an instance of A, because of the first axiom in $\mathcal{O}$ the hypertableau algorithm non-deterministically merges two individuals from the

Algorithm 2 initializeKnownAndPossibleConceptInstances($\mathcal{O}$)

Input: a consistent $\mathcal{SROIQ}$ ontology $\mathcal{O}$

```
1: A_n := buildPreModelFor(O)
2: for all a ∈ N_I^O do
3:    for all A ∈ L_{A_n}(a) do
4:       if A(a) was derived deterministically then
5:          K[A] := K[A] ∪ {a}
6:       else
7:          P[A] := P[A] ∪ {a}
8:       end if
9:    end for
10: end for
```

set $\{b_1, b_2, b_3\}$. If we assume that the individuals b_1 and b_3 are non-deterministically chosen to be merged, we get from this pre-model that b_1 is a possible r-successor of a.

We now formalize the procedure of extracting known and possible instances from the initial pre-model. In the hypertableau calculus, the following two properties hold for each ontology $\mathcal{O}$ and each constructed pre-model $\mathcal{A}_n$ for $\mathcal{O}$:

(P1) for each concept name A (role name r), each individual a (pair of individuals $\langle a, b\rangle$) in $\mathcal{A}_n$, if $A \in \mathcal{L}_{\mathcal{A}_n}(a)$ ($r \in \mathcal{L}_{\mathcal{A}_n}(\langle a, b\rangle)$) and the assertion $A(a)$ ($r(a, b)$) was derived deterministically, then it holds $\mathcal{O} \models A(a)$ ($\mathcal{O} \models r(a, b)$).

(P2) for an arbitrary individual a in $\mathcal{A}_n$ (pair of individuals $\langle a, b\rangle$ in $\mathcal{A}_n$) and an arbitrary concept name A (simple role name r), if $A \notin \mathcal{L}_{\mathcal{A}_n}(a)$ ($r \notin \mathcal{L}_{\mathcal{A}_n}(\langle a, b\rangle)$), then $\mathcal{O} \not\models A(a)$ ($\mathcal{O} \not\models r(a, b)$).

For simplicity, we assume here that equality ($\approx$) is axiomatized and $\approx$ is treated as a reflexive, symmetric, and transitive role. We use these properties to extract information from the pre-model of a satisfiable ontology $\mathcal{O}$.

Definition 9 (**Known and Possible Instances**). *Let $\mathcal{A}_n$ be a pre-model for an ontology $\mathcal{O}$. An individual a is a* known (possible) instance *of a concept name A in $\mathcal{A}_n$, denoted $a \in K_{\mathcal{A}_n}[A]$ ($a \in P_{\mathcal{A}_n}[A]$), if $A \in \mathcal{L}_{\mathcal{A}_n}(a)$ and $A(a)$ is derived deterministically (non-deterministically) in $\mathcal{A}_n$. A pair of individuals $\langle a, b\rangle$ is a* known (possible) instance *of a simple role name r in $\mathcal{A}_n$, denoted $\langle a, b\rangle \in K_{\mathcal{A}_n}(r)$, if $r \in \mathcal{L}_{\mathcal{A}_n}(\langle a, b\rangle)$ and $r(a, b)$ is derived deterministically (non-deterministically) in $\mathcal{A}_n$. The individual a is* (possibly) equal *to the individual b, written $a \in K_{\approx}[b]$ and $b \in K_{\approx}[a]$ ($a \in P_{\approx}[b]$ and $b \in P_{\approx}[a]$) if $a \approx b$ has been deterministically (non-deterministically) derived in $\mathcal{O}$.*

In the remainder, we assume that the known and possible instances are defined w.r.t. some arbitrary pre-model $\mathcal{A}_n$ for $\mathcal{O}$ and we simply write $K[A]$, $K[r]$, $K_{\approx}[a]$, $P[A]$, $P[r]$, and $P_{\approx}[a]$. Intuitively, $K[A]$ contains individuals that can safely be considered instances of the concept name A. On the other hand, the possible instances require costly consistency checks in order to decide whether they are real instances of the concept, while individuals that neither belong to $K[A]$ nor $P[A]$ can safely be assumed to be non-instances of A.

Algorithm 2 outlines a procedure to initialize the relations for known and possible concept instances. The information we extract involves the maintenance of the sets of known and possible instances for all concepts of $\mathcal{O}$. One can define a similar algorithm for initializing the known and possible instances of simple roles and for (possibly) equal individuals. In our implementation, we use a more involved procedure to only store the direct types of each individual, where a concept name A is a *direct type* of an individual a in an ontology $\mathcal{O}$ if $\mathcal{O} \models A(a)$ and there is no concept name B such that $\mathcal{O} \models B \sqsubseteq A$, $\mathcal{O} \models B(a)$ and $\mathcal{O} \not\models B \equiv A$.

Hypertableau and tableau reasoners typically do not deal with transitivity directly. In order to deal with non-simple roles, $\mathcal{O}$ is expanded with additional axioms that capture the semantics of the transitive relations before a pre-model is built. In particular, for each individual a and non-simple role r, new concepts C_a and C_a^r are introduced and the axioms $C_a(a)$ and $C_a \sqsubseteq \forall r.C_a^r$ are added to $\mathcal{O}$. The consequent application of the transitivity encoding [96] produces axioms that propagate C_a^r to each individual b that is reachable from a via an r-chain. The known and possible r-successors of a can then be determined from the C_a^r instances.

The technique presented in this paper can be used with any (hyper)tableau calculus for which properties (P1) and (P2) hold. All (hyper)tableau calculi used in practice that we are aware of satisfy property (P1). Pre-models produced by tableau algorithms as presented in the literature also satisfy property (P2); however, commonly used optimizations, such as lazy unfolding, can compromise property (P2), which we illustrate with the following example. Let us assume we have an ontology $\mathcal{O}$ containing the axioms

$$A \sqsubseteq \exists r.(C \sqcap D) \tag{4.10}$$

$$B \equiv \exists r.C \tag{4.11}$$

$$A(a) \tag{4.12}$$

It is obvious that in this ontology A is a subconcept of B (hence, $\mathcal{O} \models B(a)$) since every individual that is r-related to an individual that is an instance of the intersection of C and D is also r-related to an individual that is an instance of the concept C. However, even though the assertion $A(a)$ occurs in the ABox, the assertion $B(a)$ is not added in the pre-model when we use lazy unfolding. With lazy unfolding, instead of treating (4.11) as two disjunctions $\neg B \sqcup \exists r.C$ and $B \sqcup \forall r.(\neg C)$ as is typically done for general concept inclusion axioms, B is only lazily unfolded into its definition $\exists r.C$ once B occurs in the label of an individual. Thus, although $(\exists r.(C \sqcap D))(a)$ would be derived, this does not lead to the addition of $B(a)$.

Nevertheless, most (if not all) implemented calculi produce pre-models that satisfy at least the following weaker property:

(P3) for an arbitrary individual a in $\mathcal{A}_n$ and an arbitrary concept name A where A is primitive in $\mathcal{O}$,[1] if $A \notin \mathcal{L}_{\mathcal{A}_n}(a)$, then $\mathcal{O} \not\models A(a)$.

Hence, properties (P2) and (P3) can be used to extract (non-)instance information from pre-models. For tableau calculi that only satisfy (P3), for each non-primitive concept name A in $\mathcal{O}$ we need to add to $P[A]$ the individuals in $\mathcal{O}$ that do not include the concept A in their label.

[1] A concept A is considered primitive in $\mathcal{O}$ if $\mathcal{O}$ is unfoldable [137] and it contains no axiom of the form $A \equiv E$

The proposed technique for determining known and possible instances of concept and role names and equivalent individuals can be used in the same way with both tableau and hypertableau reasoners. Since tableau algorithms often introduce more nondeterminism than hypertableau, one might, however, find less deterministic derivations, which results in less accurate statistics.

It is worth noting that the above presented technique that creates sets of known and possible instances for concepts and roles can be seen as an approximate instance retrieval algorithm, which computes an under- (known instances) and an over- (possible instances) approximation of instances of concepts and roles.

4.2.1.1 Individual Clustering

In this section, we describe the procedure for creating clusters of individuals within an ontology $\mathcal{O}$ using a constructed pre-model $\mathcal{A}_n$ of $\mathcal{O}$. Two types of clusters are created: *concept clusters* and *role clusters*. Concept clusters contain individuals having the same concepts in their label and role clusters contain individuals with the same concept and role labels. Role clusters are divided into three categories, those that are based on the first individual of role instances, those based on the second individual and those based on both individuals.

Definition 10 (**Concept and Role Clusters**). *Let $\mathcal{O}$ be an ontology and $\mathcal{A}_n$ a pre-model for $\mathcal{O}$. We define the following two relations P_1 and P_2 that map an individual a from $\mathcal{O}$ to the roles for which a has at least one successor or predecessor, respectively:*

$$P_1(a) = \{r \mid r \in \mathcal{L}_{\mathcal{A}_n}(\langle a, b\rangle) \text{ for some } b \in N_I^{\mathcal{O}}\}$$
$$P_2(a) = \{r \mid r \in \mathcal{L}_{\mathcal{A}_n}(\langle b, a\rangle) \text{ for some } b \in N_I^{\mathcal{O}}\}$$

Based on these relations, we build three different partitions over $N_I^{\mathcal{O}}$: concept clusters CC, role successor clusters PC_1, *and* role predecessor clusters PC_2 *and we partition $N_I^{\mathcal{O}} \times N_I^{\mathcal{O}}$ into* role clusters PC_{12} *such that the clusters satisfy:*

$$\text{for each } C \in CC.(\text{for each } a_1, a_2 \in C.(\mathcal{L}_{\mathcal{A}_n}(a_1) = \mathcal{L}_{\mathcal{A}_n}(a_2)))$$
$$\text{for each } C \in PC_1.(\text{for each } a_1, a_2 \in C.(\mathcal{L}_{\mathcal{A}_n}(a_1) = \mathcal{L}_{\mathcal{A}_n}(a_2) \text{ and } P_1(a_1) = P_1(a_2)))$$
$$\text{for each } C \in PC_2.(\text{for each } a_1, a_2 \in C.(\mathcal{L}_{\mathcal{A}_n}(a_1) = \mathcal{L}_{\mathcal{A}_n}(a_2) \text{ and } P_2(a_1) = P_2(a_2)))$$
$$\text{for each } C \in PC_{12}.(\text{for each } \langle a_1, a_2\rangle, \langle a_3, a_4\rangle \in C.$$
$$(\mathcal{L}_{\mathcal{A}_n}(a_1) = \mathcal{L}_{\mathcal{A}_n}(a_3), \mathcal{L}_{\mathcal{A}_n}(a_2) = \mathcal{L}_{\mathcal{A}_n}(a_4) \text{ and } \mathcal{L}_{\mathcal{A}_n}(\langle a_1, a_2\rangle) = \mathcal{L}_{\mathcal{A}_n}(\langle a_3, a_4\rangle)))$$

We use these clusters in the next section to optimize the dynamic query ordering strategy.

4.3 Query Answering and Axiom Template Ordering

In this section, we describe two different algorithms (a static and a dynamic one) for ordering the axiom templates of a query based on some costs and then we deal with the formulation of these costs. We first introduce the abstract graph representation of a query q by means of a labeled graph G_q on which we define the computed statistical costs.

Definition 11 (**Query Join Graph**). *A* query join graph G_q *for a query* q *is a tuple* (V, E, E_L)*, where*

- $V = q$ *is a set of vertices (one for each axiom template);*
- $E \subseteq V \times V$ *is a set of edges; such that* $\langle \mathsf{at}_1, \mathsf{at}_2 \rangle \in E$ *if* $\mathsf{Var}(\mathsf{at}_1) \cap \mathsf{Var}(\mathsf{at}_2) \neq \emptyset$ *and* $\mathsf{at}_1 \neq \mathsf{at}_2$*;*
- E_L *is a function that assigns a set of variables to each* $\langle \mathsf{at}_1, \mathsf{at}_2 \rangle \in E$ *such that* $E_L(\mathsf{at}_1, \mathsf{at}_2) = \mathsf{Var}(\mathsf{at}_1) \cap \mathsf{Var}(\mathsf{at}_2)$*.*

In the remainder, we use G_q for the query join graph of q.

Our goal is to find a query execution plan, which determines the evaluation order for axiom templates in q. Since the number of possible execution plans is of order $|q|!$, the ordering task quickly becomes impractical. In the following, we focus on greedy algorithms for determining an execution order, which prune the search space considerably. Roughly speaking, we proceed as follows: We define a cost function, which consists of two components (i) an estimate for the costs of the reasoning tasks needed for the evaluation of an axiom template and (ii) an estimate for the result size, i.e., the number of results that the evaluation of an axiom template will incur. Both components are combined to induce an order among axiom templates. In this paper, we simply build the sum of the two cost components, but different combinations such as a weighted sum of the two values could also be used. For the query plan construction we distinguish *static* from *dynamic planning*. For the former, we start constructing the plan by adding a minimal template according to the order. Variables from this template are then considered bound, which changes the cost function and might induce a different order among the remaining axiom templates. Considering the updated order, we again select the minimal axiom template that is not yet in the plan and update the costs. This process continues until the plan contains all templates. Once a complete plan has been determined the templates are evaluated. The dynamic case differs in that after selecting a template for the plan, we immediately determine the solutions for the chosen template, which are then used to update the cost function. While this yields accurate cost estimates, it can be very costly when all solutions are considered for updating the cost function. Sampling techniques can be used to only test a subset of the solutions, but we show in Section 7 that random sampling, i.e., randomly choosing a percentage of the individuals from the so far computed solutions, is not adequate. For this reason, we propose an alternative sampling approach that is based on the use of the previously described individual clusters.

We first briefly describe how we can use the sets of known and possible concept and role instances and the sets of known and possible equivalent individuals, which have been constructed during the initial ontology consistency check, as explained in Section 4.2.1, for optimizing the evaluation of some simple axiom templates. The method callSpecificReasonerTask in Algorithm 1, line 6 from Section 4.1, has been implemented to evaluate such axiom templates more efficiently using these cached sets.

1. For concept (role) atoms of the form $A(?x)$ ($r(?x, ?y)$) the set of known instances of the concept A (role r) are immediately returned as answers by the

method callSpecificReasonerTask, while only the individuals in the set of possible instances that, when used to instantiate the atoms, lead to entailed axioms, are returned as answers by the method.

2. For role atoms of the form $r(a, ?x)$ ($r(?x, a)$), the sets of known and possible instances of r are again exploited to extract known and possible r-successors (predecessors) of a. Known successors (predecessors) of a are immediately returned as answers by the method callSpecificReasonerTask, while only those possible successors (predecessors) that when used to instantiate the atoms, lead to entailed axioms, are returned as answers by the method.

3. For the evaluation of templates of the form $?z(?x)$ ($?z(?x, ?y)$, $?z(a, ?x)$, $?z(?x, a)$) the answer is the union of the answers coming from the iteration over all concept (role) names appearing in the queried ontology; in each such iteration, a concept A (role s) is considered, i.e., we have $A(?x)$ ($s(?x, ?y)$, $s(a, ?x)$, $s(?x, a)$), and the procedure followed in case 1 (cases 1 and 2) above can be applied to find the answers of the concept (role) instantiated template.

4. For templates of the form $?z(a)$ ($?z(a, b)$) the concepts (roles) for which a ($\langle a, b\rangle$) is a known instance (pair of instances) are returned, while from the concepts for which a ($\langle a, b\rangle$) is a possible instance (pair of instances) only those which when used to instantiate the templates lead to entailed axioms are returned as answers.

5. Similarly, for atoms of the form $a \approx ?x$ or $?x \approx a$ the known and possible equivalent sets for the individual a can be used to either directly (from the set of known equivalent individuals) or after some consistency checks (using the set of possible equivalent individuals) retrieve the answers of the atom evaluation.

6. For atoms of the form $?x \approx ?y$ the answer is the union of the answers coming from the iteration over all individuals appearing in the queried ontology; in each such iteration, an individual a is considered, i.e., we have $?x \approx a$ or $a \approx ?x$ and the procedure followed in case 5 can be applied to find the answers of the individual instantiated atom.

7. For (Boolean) atoms of the form $A(a)$, $r(a, b)$, $a \approx b$ the sets of known and possible instances are again used to decide whether the axiom is entailed with (set of known instances and known equivalent individuals) or without (set of possible instances and possible equivalent individuals) a consistency check.

Assuming that the concept and role hierarchies are precomputed before the beginning of query evaluation and cached in the reasoner's internal structures, templates of the form $c_{t_1} \sqsubseteq c_{t_2}$ and $r_{t_1} \sqsubseteq r_{t_2}$ are evaluated by performing look-ups in the cached hierarchies.

We now present an example to make the difference between static and dynamic planning clearer and justify why dynamic ordering can be beneficial in our setting.

Example 6. *Let $\mathcal{O}$ be an ontology and $q = \{A(?x), r(?x, ?y), B(?y)\}$ a conjunctive instance query over $\mathcal{O}$. Suppose that for the known and possible instances of the*

query concepts and roles we have

$$K[A] = \{a\} \qquad K[r] = \emptyset \qquad K[B] = \{b\}$$
$$P[A] = \{c, e\} \qquad P[r] = \{\langle c, d\rangle, \langle e, f\rangle\} \qquad P[B] = \{f, g, h\}$$

and let us assume that the possible instances of A, B and r are, in fact, real instances (note that we do not have this information from the beginning). Please have in mind that the possible instances of concepts or roles are more costly to evaluate than the known instances since they require expensive consistency checks in order to decide whether they are real instances.

According to static planning, an ordering for query atoms is first determined. In particular, the atom $r(?x, ?y)$ is chosen first since it has the least number of known and possible instances ($|K[r]| = 0$ and $|P[r]| = 2$ versus $|K[A]| = 1$ and $|P[A]| = 2$ and $|K[B]| = 1$ and $|P[B]| = 3$). Then the atom $A(?x)$ is chosen since it has less known and possible instances than $B(?y)$, i.e., $|K[A]| = 1$ and $|P[A]| = 2$ versus $|K[B]| = 1$ and $|P[B]| = 3$ for $B(?y)$. Hence, the chosen execution plan in static planning is $P = (r(?x, ?y), A(?x), B(?y))$. Afterwards, the query is evaluated according to the chosen execution plan, i.e., the atom $r(?x, ?y)$ is evaluated first, which gives the solution mappings

$$\Omega_1 = \{\{?x \mapsto c, ?y \mapsto d\}, \{?x \mapsto e, ?y \mapsto f\}\}.$$

This requires 2 consistency checks for the 2 possible instances of r. Afterwards, we check which of the $?x$ mappings, c and e, are known or possible instances of A. Since both c and e are possible instances, we check whether they are real instances of A (this requires 2 consistency checks). Hence, the solution mappings are

$$\Omega_2 = \Omega_1 = \{\{?x \mapsto c, ?y \mapsto d\}, \{?x \mapsto e, ?y \mapsto f\}\}.$$

In the end, we check which of the $?y$ mappings, d and f, are known or possible instances of B. For the only possible instance, f, we find after one consistency check that f is indeed an instance of B. Hence, the solution mappings for q over $\mathcal{O}$ are

$$\Omega_q^{\mathcal{O}} = \{\{?x \mapsto e, ?y \mapsto f\}\}$$

and finding the solution required 5 consistency checks.

According to dynamic planning, an ordering is determined while we evaluate the query. For the same reasons as before, the atom $r(?x, ?y)$ is chosen to be evaluated first and the solution mappings are, as before,

$$\Omega_1 = \{\{?x \mapsto c, ?y \mapsto d\}, \{?x \mapsto e, ?y \mapsto f\}\}$$

requiring 2 consistency checks. We afterwards check which of the $?y$ mappings, d and f, are known or possible instances of B. Note that this only requires a look-up since if we find d or f to be among the possible instances, we do not check whether the individual is indeed an instance or not. Here only f is a possible instance. We also check which of the $?x$ mappings, c and e, are known or possible instances of A. Here, both c and e are possible instances, i.e., we have 2 relevant possible instances

for $A(?x)$ and 1 for $B(?y)$. Hence, the atom $B(?y)$ is chosen to be evaluated next, resulting in the solution sequence

$$\Omega_2 = \{\{?x \mapsto e, ?y \mapsto f\}\}$$

for the (partial) execution plan $(r(?x, ?y), B(?y))$, requiring 1 consistency check. In the end, we check whether the $?x$ mapping, e, is a known or possible instance of A. Since e is a possible instance, we check whether it is a real instance (this requires 1 consistency check). Hence, the solution mappings for q are

$$\Omega_q^{\mathcal{O}} = \{\{?x \mapsto e, ?y \mapsto f\}\},$$

which have been found by performing 4 consistency checks, one less than in the static case.

Note that in dynamic ordering we perform less checks than in static ordering, since in this case we can exploit the results of joins of query atoms and more information regarding the possible instances of atoms (i.e., which of them are real instances), which is determined as a result of evaluating the atoms while ordering them.

We now make the process of query plan construction more precise, but we leave the exact details of defining the cost function and the ordering it induces to later.

Definition 12 (**Static and Dynamic Ordering**)**.** *A* static (dynamic) cost function w.r.t. q over $\mathcal{O}$ *is a function $s\colon q \times 2^{\mathsf{Var}(q)} \to \mathbb{R} \times \mathbb{R}$ ($d\colon q \times 2^{\Gamma_q^{\mathcal{O}}} \to \mathbb{R} \times \mathbb{R}$), where with $\Gamma_q^{\mathcal{O}}$ we denote the set of compatible mappings for q over $\mathcal{O}$. The two costs $\langle \mathsf{Ec}_{\mathsf{at}}^s, \mathsf{Rs}_{\mathsf{at}}^s \rangle$ ($\langle \mathsf{Ec}_{\mathsf{at}}^d, \mathsf{Rs}_{\mathsf{at}}^d \rangle$) for an axiom template $\mathsf{at} \in q$ are combined to yield a* static ordering $\preceq_s$ (dynamic ordering $\preceq_d$)*, which is a total order over the axiom templates of q such that, for $\mathsf{at}, \mathsf{at}' \in q$, we say that $\mathsf{at} \preceq_s \mathsf{at}'$ ($\mathsf{at} \preceq_d \mathsf{at}'$) iff $\mathsf{Ec}_{\mathsf{at}}^s + \mathsf{Rs}_{\mathsf{at}}^s \leq \mathsf{Ec}_{\mathsf{at}'}^s + \mathsf{Rs}_{\mathsf{at}'}^s$ ($\mathsf{Ec}_{\mathsf{at}}^d + \mathsf{Rs}_{\mathsf{at}}^d \leq \mathsf{Ec}_{\mathsf{at}'}^d + \mathsf{Rs}_{\mathsf{at}'}^d$).*

An execution plan *for q is a duplicate-free sequence of axiom templates from q. The* initial execution plan *is the empty sequence and a* complete execution plan *is a sequence containing all templates of q. Let $P_i = (\mathsf{at}_1, \ldots, \mathsf{at}_i)$ with $i < |q|$ be an execution plan for q with query join graph $G_q = (V, E, E_L)$. The set of* bound variables *of at_i within P_i for $i > 1$ is $V_b(\mathsf{at}_i) = \mathsf{Var}(\mathsf{at}_i) \cap \mathsf{Var}(\{\mathsf{at}_1, \ldots, \mathsf{at}_{i-1}\})$ else $\mathsf{Var}(\mathsf{at}_i) = \emptyset$. Let C_q be the set of complex axiom templates in q. We next define which axiom templates can be used to extend an incomplete execution plan. Let at be an axiom template in P_i, the set $suc_i(\mathsf{at})$ contains the axiom templates that are connected to at and not yet in P_i, i.e., $suc_i(\mathsf{at}) = \{\mathsf{at}' \in q \mid \langle \mathsf{at}, \mathsf{at}' \rangle \in E, \mathsf{at}' \notin \{\mathsf{at}_1, \ldots \mathsf{at}_i\}\}$. Based on this, we define the set of* connected successor axiom templates for P_i *as $S_i = \{\mathsf{at} \mid \mathsf{at}' \in \{\mathsf{at}_1, \ldots, \mathsf{at}_i\}$ and $\mathsf{at} \in suc_i(\mathsf{at}')\}$. We further allow for including axiom templates that are only connected to a complex axiom template from S_i and define the* potential next templates q_i for P_i w.r.t. G_q *as $q_i = q$ if P_i is the initial execution plan and otherwise*

$$q_i = S_i \cup \bigcup_{\mathsf{at} \in C_q \cap S_i} suc_i(\mathsf{at}).$$

Given P_i, the static (dynamic) ordering now induces an execution plan $P_{i+1} = (\mathsf{at}_1, \ldots, \mathsf{at}_i, \mathsf{at}_{i+1})$ with $\mathsf{at}_{i+1} \in q_i$ and $\mathsf{at}_{i+1} \preceq_s \mathsf{at}$ ($\mathsf{at}_{i+1} \preceq_d \mathsf{at}$) for each $\mathsf{at} \in q_i$ such that $\mathsf{at} \neq \mathsf{at}_{i+1}$.

Note that according to the above definition, for P_i an execution plan, it can be the case that q_i contains templates that are assigned the same minimal cost by the cost function. In such case, one can choose any of these templates to add to P_i. Moreover, according to the above definition for the case of queries containing only simple axiom templates we have that, for $i > 0$, the set of potential next templates only contains templates that are connected to a template that is already in the plan since unconnected templates cause an unnecessary blowup of the number of intermediate results. For queries with complex templates the set of potential next axiom templates can additionally contain templates that do not share common variables with any template that is already in the plan. This different handling of queries with complex templates is reasonable since, before evaluating a complex axiom template that requires many consistency checks, we want to reduce the number of candidate bindings, by first evaluating other simple (cheaper) templates that bind variables which appear in the complex one.

Example 7. *Let $\mathcal{O}$ be an ontology and $q = \{?x \sqsubseteq A, ?y \sqsubseteq r, B \sqsubseteq \exists ?y.?x\}$ a query. Assuming that systems usually precompute the concept and role hierarchies before they accept queries, the evaluation of the first two templates, i.e., $?x \sqsubseteq A$ and $?y \sqsubseteq r$, require cheap cache look-ups, whereas the axiom template $B \sqsubseteq \exists ?y.?x$, requires costly consistency checks. Hence, it is reasonable to first evaluate the first two (cheap) templates to reduce the mappings for $?x$ and $?y$ and then evaluate the third (expensive) template, by checking which of the reduced mappings yield an entailed axiom.*

An example that shows the actual gain we get from handling the ordering of complex axiom templates in this way is presented in Chapter 7.

Let $n = |q|$ and $P_n = (\mathsf{at}_1, \ldots, \mathsf{at}_n)$ be a complete execution plan for q over $\mathcal{O}$ determined by static ordering. The procedure to find the solution mappings $\Omega_q^{\mathcal{O}}$ for P_n is recursively defined as follows: Initially, our solution set contains only the identity mapping $\Omega_0 = \{\mu_0\}$, which does not map any variable to any value. Assuming that we have evaluated the sequence $P_i = (\mathsf{at}_1, \ldots, \mathsf{at}_i)$, $i < n$ and we have found the set of solution mappings Ω_i, in order to find the solution mappings Ω_{i+1} of P_{i+1}, we use specific reasoning tasks to extend the mappings in Ω_i to cover the new variables of at_{i+1} if at_{i+1} is a simple axiom template or the entailment check service of reasoners if at_{i+1} does not contain new variables or if at_{i+1} is a complex axiom template. In dynamic planning the difference is that the execution plan construction is interleaved with query evaluation. In particular, let $n = |q|$ and $P_i = (\mathsf{at}_1 \ldots \mathsf{at}_i)$ with $i < n$ be a (partial) execution plan for q determined by dynamic ordering and let Ω_i be the solution mappings of P_i. In order to find P_{i+1} we extend P_i with a new template, at_{i+1}, from q, i.e., $P_{i+1} = (\mathsf{at}_1, \ldots \mathsf{at}_{i+1})$, which, according to the dynamic cost function, has the minimal cost among the potential next templates q_i for P_i. The dynamic cost function assigns costs to templates at iteration $i + 1$ taking into account the solution mappings Ω_i. We afterwards evaluate the atom at_{i+1}, i.e., we find the solution mappings Ω_{i+1} of P_{i+1} by extending the solution mappings Ω_i of P_i in the same way as in the static case. In Chapter 6 in Algorithm 4, we show the complete procedure we follow to answer a query.

We now define the cost functions s and d more precisely, which estimate the cost of the required reasoner operations (first component) and the estimated result output size (second component) of evaluating an axiom template. The intuition

behind the estimated value of the reasoner operation costs is that the evaluation of possible instances is much more costly than the evaluation of known instances since possible instances require expensive consistency checks whereas known instances require cheap cache look-ups. The estimated result size takes into account the number of known and possible instances and the probability that possible instances are actual instances.

The time needed for an entailment check can change considerably between ontologies and even within an ontology (depending on the involved concepts, roles and individuals). In order to more accurately determine the entailment cost we use different entailment cost values depending on whether the template under consideration is a template of the form $c_{t_1}(t)$, $r_{t_1}(t, t')$, $t \approx t'$, $c_{t_1} \sqsubseteq c_{t_2}$, $r_{t_1} \sqsubseteq r_{t_2}$ or a complex axiom template, where c_{t_1}, c_{t_2} are concept terms, r_{t_1}, r_{t_2} are role terms and t, t' are individual terms. In the following we write C_L to denote the cost of a cache look-up in the internal structures of the reasoner, C_E as a placeholder for the relevant entailment cost value and P_{IS} for the possible instance success, i.e, the estimated percentage of possible instances that are actual instances. The costs C_L and C_E are determined by recording the average time of previously performed look-ups and entailment checks for the queried ontology, e.g., during the initial consistency check, classification, or for previous queries. The possible instance success, P_{IS}, was determined by testing several ontologies and checking how many of the initial possible instances were real ones, which was around 50% in nearly all ontologies.

Apart from the relations for the known and possible instances from Section 4.2.1, we use the following auxiliary relations:

Definition 13 (**Successor and Predecessor Relations**). *Let r be a role and a an individual. We define* $\mathsf{sucK}[r]$ *and* $\mathsf{preK}[r]$ *as the set of individuals with known r-successors and r-predecessors, respectively:*

$$\mathsf{sucK}[r] := \{a \mid \exists b.\langle a, b\rangle \in K[r]\} \qquad \textit{and} \qquad \mathsf{preK}[r] := \{a \mid \exists b.\langle b, a\rangle \in K[r]\}.$$

Similarly, we define $\mathsf{sucK}[r, a]$ *and* $\mathsf{preK}[r, a]$ *as the known r-successors of a and the known r-predecessors of a, respectively:*

$$\mathsf{sucK}[r, a] := \{b \mid \langle a, b\rangle \in K[r]\} \qquad \textit{and} \qquad \mathsf{preK}[r, a] := \{b \mid \langle b, a\rangle \in K[r]\}.$$

We analogously define the functions $\mathsf{sucP}[r]$, $\mathsf{preP}[r]$, $\mathsf{sucP}[r, a]$, *and* $\mathsf{preP}[r, a]$ *by replacing $K[r]$ with $P[r]$ in the above definitions.*

Next, we define the cost functions for the case of conjunctive instance queries, i.e., queries containing only query atoms. In Section 4.3.2 we extend the cost functions to deal with general queries.

4.3.1 Cost Functions for Conjunctive Instance Queries

The static cost function s takes two components as input: a query atom and a set containing the variables of the query atom that are considered bound. The function returns a pair of real numbers for the reasoning cost and the result size for the query atom.

Initially, all variables are unbound and we use the number of known and possible instances or successors/predecessors to estimate the number of required look-ups

and consistency checks for evaluating the query atom and for the resulting number of mappings. For an input of the form $\langle A(?x), \emptyset\rangle$ or $\langle r(?x, ?y), \emptyset\rangle$ the resulting pair of real numbers for the computational cost and the estimated result size is computed as

$$\langle |K[\mathsf{at}]| \cdot d \cdot C_L + |P[\mathsf{at}]| \cdot d \cdot C_E, |K[\mathsf{at}]| + P_{IS} \cdot |P[\mathsf{at}]|\rangle,$$

where at denotes the predicate of the query atom (A or r) and the factor d represents the depth of the concept for $\mathsf{at} = A(?x)$ or role for $\mathsf{at} = r(?x, ?y)$ hierarchy. We use this factor since we only store the direct types of each individual (roles of which individuals are instances) and, in order to find the instances of a concept (role), we may need to check all its sub-concepts (sub-roles) for known or possible instances. If the query atom is a role atom with a constant in the first place, i.e., the input to the cost function is of the form $\langle r(a, ?x), \emptyset\rangle$, we use the relations for known and possible successors to estimate the computational cost and result size:

$$\langle |\mathsf{sucK}[r, a]| \cdot d \cdot C_L + |\mathsf{sucP}[r, a]| \cdot d \cdot C_E, |\mathsf{sucK}[r, a]| + P_{IS} \cdot |\mathsf{sucP}[r, a]|\rangle.$$

Analogously, we use preK and preP instead of sucK and sucP for an input of the form $\langle r(?x, a), \emptyset\rangle$. Finally, if the atom contains only constants, i.e., the input to the cost function is of the form $\langle A(a), \emptyset\rangle, \langle r(a, b), \emptyset\rangle$, the function returns $\langle d \cdot C_L, 1\rangle$ if the individual is a known instance of the concept or role, $\langle d \cdot C_E, P_{IS}\rangle$ if the individual is a possible instance and $\langle d \cdot C_L, 0\rangle$ otherwise, i.e., if the individual is a known non-instance.

For equality atoms of the form $?x \approx ?y$, $a \approx ?x$, $?x \approx a$ or $a \approx b$, we again exploit information from the initial pre-model as described in Section 4.2.1. Based on the cardinality of $K_{\approx}[a]$ and $P_{\approx}[a]$, we can define cost functions for the different cases of query atoms and bound variables. For inputs of the form $\langle ?x \approx a, \emptyset\rangle$ and $\langle a \approx ?x, \emptyset\rangle$, the cost function is defined as:

$$\langle |K_{\approx}[a]| \cdot C_L + |P_{\approx}[a]| \cdot C_E, |K_{\approx}[a]| + P_{IS} \cdot |P_{\approx}[a]|\rangle.$$

For inputs of the form $\langle ?x \approx ?y, \emptyset\rangle$, the cost function is computed as:

$$\left\langle \sum_{a \in N_I^{\mathcal{O}}} (|K_{\approx}[a]| \cdot C_L + |P_{\approx}[a]| \cdot C_E)/2, \sum_{a \in N_I^{\mathcal{O}}} (|K_{\approx}[a]| + P_{IS} \cdot |P_{\approx}[a]|)/2 \right\rangle$$

For inputs of the form $\langle a \approx b, \emptyset\rangle$, the function returns $\langle C_L, 1\rangle$ if $b \in K_{\approx}[a]$, $\langle C_E, P_{IS}\rangle$ if $b \in P_{\approx}[a]$, and $\langle C_L, 0\rangle$ otherwise (i.e., b is not equivalent to a).

After determining the cost of an initial query atom, at least one variable of a consequently considered atom is bound, since during the query plan construction we move over atoms sharing a common variable and we assume that the query is connected. We now define the cost functions for atoms with at least one variable bound. We make the assumption that atoms with unbound variables are more costly to evaluate than atoms with all their variables bound. For a query atom $r(?x, ?y)$ with only $?x$ bound, i.e., function inputs of the form $\langle r(?x, ?y), \{?x\}\rangle$, we use the average number of known and possible successors of the role to estimate the computational cost and result size:

$$\left\langle \frac{|K[r]|}{|\mathsf{sucK}[r]|} \cdot d \cdot C_L + \frac{|P[r]|}{|\mathsf{sucP}[r]|} \cdot d \cdot C_E, \frac{|K[r]|}{|\mathsf{sucK}[r]|} + \frac{|P[r]|}{|\mathsf{sucP}[r]|} \cdot P_{IS} \right\rangle$$

In case only $?y$ in $r(?x, ?y)$ is bound, we use the predecessor functions preK and preP instead of sucK and sucP. Note that we now work with an estimated average number of successors (predecessors) for *one individual*.

For atoms with all their variables bound, we use formulas that are comparable to the ones above for an initial plan, but normalized to estimate the values for one individual. For an input query atom of the form $A(?x)$ with $?x$ a bound variable we use

$$\left\langle \frac{|K[A]| \cdot d \cdot C_L + |P[A]| \cdot d \cdot C_E}{|N_I^{\mathcal{O}}|}, \frac{|K[A]| + P_{IS} \cdot |P[A]|}{|N_I^{\mathcal{O}}|} \right\rangle$$

Such a simple normalization is not always accurate, but leads to good results in most cases as we show in Chapter 7. Similarly, we normalize the formulas for role atoms of the form $r(?x, ?y)$ such that $\{?x, ?y\}$ is the set of bound variables of the atom. The two cost components for these atoms are computed as

$$\left\langle \frac{|K[r]| \cdot d \cdot C_L + |P[r]| \cdot d \cdot C_E}{|N_I^{\mathcal{O}}| \cdot |N_I^{\mathcal{O}}|}, \frac{|K[r]| + P_{IS} \cdot |P[r]|}{|N_I^{\mathcal{O}}| \cdot |N_I^{\mathcal{O}}|} \right\rangle$$

For role atoms with a constant and a bound variable, i.e., atoms of the form $r(a, ?x)$ ($r(?x, a)$) with $?x$ a bound variable, we use $\mathsf{sucK}[r, a]$ and $\mathsf{sucP}[r, a]$ ($\mathsf{preK}[r, a]$ and $\mathsf{preP}[r, a]$) instead of $K[r]$ and $P[r]$ in the above formulas and we normalize by $|N_I^{\mathcal{O}}|$.

Similarly, we normalize the cost functions for inputs with equality atoms and bound variables, depending on whether the atoms contain one or two bound variables. For inputs of the form $\langle ?x \approx a, \{?x\}\rangle$, $\langle a \approx ?x, \{?x\}\rangle$, we divide the cost function components for inputs of the form $\langle ?x \approx a, \emptyset\rangle$ and $\langle a \approx ?x, \emptyset\rangle$ by $|N_I^{\mathcal{O}}|$. For an input of the form $\langle ?x \approx y, \{?x, ?y\}\rangle$, we divide the cost function components for input of the form $\langle ?x \approx ?y, \emptyset\rangle$ by $|N_I^{\mathcal{O}}| \cdot |N_I^{\mathcal{O}}|$. For inputs of the form $\langle ?x \approx ?y, \{?x\}\rangle$, and $\langle ?x \approx ?y, \{?y\}\rangle$, we divide the cost function components for input of the form $\langle ?x \approx ?y, \emptyset\rangle$ by $|N_I^{\mathcal{O}}|$.

The dynamic cost function d is based on the static function s, but only uses the first equations, where the atom contains only unbound variables or constants. The function takes a pair $\langle \mathsf{at}, \Omega\rangle$ as input, where at is a query atom and Ω is the set of solution mappings for the atoms that have already been evaluated, and returns a pair of real numbers using matrix addition as follows:

$$d(\mathsf{at}, \Omega) = \sum_{\mu \in \Omega} s(\mu(\mathsf{at}), \emptyset)$$

When sampling techniques are used, we compute the costs for each of the potential next atoms for an execution plan by only considering one individual of each relevant cluster. Which cluster is relevant depends on the query atom for which we compute the cost function and the previously computed bindings. For instance, if we compute the cost of a role atom $r(?x, ?y)$ and we have already determined bindings for $?x$, we use the role successor cluster PC_1. Among the $?x$ bindings, we then just check the cost for one binding per cluster and assign the same cost to all other $?x$ bindings of the same cluster.

Example 8. *Let q be a conjunctive instance query that contains the atom $A(?x)$ and let us assume that we have to find the cost (using the dynamic function) of $A(?x)$ within an execution plan for q. We further assume that from the evaluation of previous query atoms in the plan we have already determined a set of intermediate*

Table 4.1: *Query Ordering Example*

	already executed	current atom at	$K[\mathsf{at}]$	$P[\mathsf{at}]$	real from $P[\mathsf{at}]$
1		$A(?x)$	200	350	200
2		$r(?x, ?y)$	200	200	50
3		$B(?y)$	700	600	400
4	$r(?x, ?y)$	$A(?x)$	100	150	100
5	$r(?x, ?y)$	$B(?y)$	50	50	40
6	$r(?x, ?y)$, $B(?y)$	$A(?x)$	45	35	25
7	$r(?x, ?y)$, $A(?x)$	$B(?y)$	45	40	25

solutions Ω *with the mappings* $a, b,$ *or* c *for* $?x$ *and that* $a, b,$ *and* c *belong to the same concept cluster. According to* dynamic *ordering we need to find the cost of each* instantiated *atom using the* static *cost function, i.e.,*

$$d(A(?x), \Omega) = s(A(a), \emptyset) + s(A(b), \emptyset) + s(A(c), \emptyset).$$

If we additionally use cluster based sampling, we find the cost for only one individual of each cluster, let us say a*, and then assign the same cost to all other individuals from the cluster which are mappings for* $?x$ *in* Ω*. Hence, the cost of the atom* $A(?x)$ *when sampling is used, is computed as*

$$d(A(?x), \Omega) = 3 \cdot s(A(a), \emptyset)$$

avoiding the computation of $s(A(b), \emptyset)$ *and* $s(A(c), \emptyset)$*.*

An example that is similar to Example 6 (but with a greater number of instances) and shows how ordering is achieved by the use of the defined static and dynamic functions is shown below. We assume that q is a query consisting of the three query atoms: $A(?x)$, $r(?x, ?y)$, $B(?y)$. Table 4.1 gives information about the known and possible instances of these atoms within a sequence. The second column shows already executed sequences $P_{i-1} = (\mathsf{at}_1, \ldots, \mathsf{at}_{i-1})$ for the atoms of q. Column 3 gives the current atom at_i and column 4 (5) gives the number of mappings to known (possible) instances of at that satisfy at the same time the atoms $(\mathsf{at}_1, \ldots, \mathsf{at}_{i-1})$ from column 2. Column 6 gives the number of real instances from the possible instances for the current atom. For example, row 4 says that we have evaluated the atom $r(?x, ?y)$ and, in order to evaluate $A(?x)$, we only consider those 100 known and 150 possible instances of A that are also mappings for $?x$. We further assume that we have 10,000 individuals in our ontology $\mathcal{O}$. We now explain, using the example, how the above described formulas work. We assume that $C_L \leq C_E$, which is always the case since a cache look-up is less expensive than a consistency check and that the C_E values are the same for all query concepts and roles. For ease of presentation, we further do not consider the factor for the depth of the concept or role hierarchies. In both techniques (static and dynamic) the atom $r(?x, ?y)$ is chosen first since it has the least number of possible instances (200) while it has the same (or smaller) number of known instances (200) as the other atoms (μ_0 is the initial solution mapping that does not map any variable):

$$\begin{aligned} s(r(?x, ?y), \emptyset) = d(r(?x, ?y), \{\mu_0\}) &= \langle 200 \cdot C_L + 200 \cdot C_E, 200 + P_{IS} \cdot 200 \rangle, \\ s(A(?x), \emptyset) = d(A(?x), \{\mu_0\}) &= \langle 200 \cdot C_L + 350 \cdot C_E, 200 + P_{IS} \cdot 350 \rangle, \\ s(B(?y), \emptyset) = d(B(?y), \{\mu_0\}) &= \langle 700 \cdot C_L + 600 \cdot C_E, 700 + P_{IS} \cdot 600 \rangle. \end{aligned}$$

In the case of static ordering, the atom $A(?x)$ is chosen after $r(?x, ?y)$ since A has less possible (and known) instances than B (350 versus 600):

$$s(A(?x), \{?x\}) = \left\langle \frac{200}{10,000} \cdot C_L + \frac{350}{10,000} \cdot C_E, \frac{200 + 350 \cdot P_{IS}}{10,000} \right\rangle,$$
$$s(B(?y), \{?y\}) = \left\langle \frac{700}{10,000} \cdot C_L + \frac{600}{10,000} \cdot C_E, \frac{700 + 600 \cdot P_{IS}}{10,000} \right\rangle.$$

Hence, the order of evaluation in this case is $P = (r(?x, ?y), A(?x), B(?y))$ leading to 200 (row 2) + 150 (row 4) + 40 (row 7) entailment checks. In the dynamic case, after the evaluation of $r(?x, ?y)$, which gives a set of solutions Ω_1, the atom $B(?y)$ has fewer known and possible instances (50 known and 50 possible) (row 5) than the atom $A(?x)$ (100 known and 150 possible) (row 4) and, hence, a lower cost:

$$d(B(?y), \Omega_1) = \langle 50 \cdot C_L + 150 \cdot C_L + 50 \cdot C_E, 50 + 0 + 50 \cdot P_{IS} \rangle,$$
$$d(A(?x), \Omega_1) = \langle 100 \cdot C_L + 0 \cdot C_L + 150 \cdot C_E, 100 + 0 + 150 \cdot P_{IS} \rangle.$$

Note that applying a solution $\mu \in \Omega_1$ to $B(?y)$ ($A(?x)$) results in a query atom with a constant in place of $?y$ ($?x$). For $B(?y)$, it is the case that out of the 250 r-instances, 200 can be handled with a look-up (50 turn out to be known instances and 150 turn out not to be instances of B), while 50 require an entailment check. Similarly, when considering $A(?x)$, we need 100 look-ups and 150 entailment checks. Note that we assume the worst case in this example, i.e., that all values that $?x$ and $?y$ take are different. Therefore, the atom $B(?y)$ is chosen next, leading to the execution of the query atoms in the order $P = (r(?x, ?y), B(?y), A(?x))$ and the execution of 200 (row 2) + 50 (row 5) + 35 (row 6) entailment checks.

4.3.2 Cost Functions for General Queries

We now explain how we order the remaining simple and complex axiom templates. We again use statistics from the reasoner, whenever these are available. In case the reasoner cannot give estimates, one can still work with statistics computed from explicitly stated information or use upper bounds to estimate the reasoner costs and the result size of axiom templates.

We first consider a general concept assertion axiom template. Let $K_B[a]$ be the concepts of which a is a known instance, $P_B[a]$ the concepts of which a is a possible instance. These sets are computed from the sets of known and possible instances of concepts. For an input of the form $\langle ?x(a), \emptyset \rangle$ the cost function is defined as

$$\langle |K_B[a]| \cdot d \cdot C_L + |P_B[a]| \cdot d \cdot C_E, |K_B[a]| + P_{IS} \cdot |P_B[a]| \rangle,$$

For an input of the form $\langle ?x(?y), \emptyset \rangle$, the cost function is defined as

$$\left\langle \sum_{B \in N_C^{\mathcal{O}}} (|K[B]| \cdot d \cdot C_L + |P[B]| \cdot d \cdot C_E), \sum_{B \in N_C^{\mathcal{O}}} (|K[B]| + P_{IS} \cdot |P[B]|) \right\rangle$$

For inputs of the form $\langle ?x(a), \{?x\} \rangle$ and $\langle ?x(?y), \{?x, ?y\} \rangle$, we normalize the above functions by $|N_C^{\mathcal{O}}|$ and $|N_I^{\mathcal{O}}| \cdot |N_C^{\mathcal{O}}|$ respectively. For inputs of the form $\langle ?x(?y), \{?x\} \rangle$ and $\langle ?x(?y), \{?y\} \rangle$ we normalize the function for inputs of the form $\langle ?x(?y), \emptyset \rangle$ by $|N_C^{\mathcal{O}}|$ and $|N_I^{\mathcal{O}}|$ respectively.

For general role assertion axiom templates, there are several cases of cost functions depending on the bound variables. We next define the cost functions for some cases. The cost functions for the other cases can similarly be defined. For an input of the form $\langle ?z(?x, ?y), \emptyset \rangle$, the cost function is defined as :

$$\left\langle \sum_{r \in N_R^{\mathcal{O}}} (|K[r]| \cdot d \cdot C_L + |P[r]| \cdot d \cdot C_E), \sum_{r \in N_R^{\mathcal{O}}} (|K[r]| + P_{IS} \cdot |P[r]|) \right\rangle.$$

For inputs of the form $\langle ?z(a, ?y), \emptyset \rangle$, the cost function is defined as:

$$\left\langle \sum_{r \in N_R^{\mathcal{O}}} (|\mathsf{sucK}[r, a]| \cdot d \cdot C_L + |\mathsf{sucP}[r, a]| \cdot d \cdot C_E), \sum_{r \in N_R^{\mathcal{O}}} (|\mathsf{sucK}[r, a]| + P_{IS} \cdot |\mathsf{sucP}[r, a]|) \right\rangle.$$

For an input of the form $\langle ?z(?x, ?y), \{?z\} \rangle$, the cost function is defined as:

$$\left\langle \sum_{r \in N_R^{\mathcal{O}}} \frac{|K[r]| \cdot d \cdot C_L + |P[r]| \cdot d \cdot C_E}{|N_R^{\mathcal{O}}|}, \sum_{r \in N_R^{\mathcal{O}}} \frac{|K[r]| + P_{IS} \cdot |P[r]|}{|N_R^{\mathcal{O}}|} \right\rangle.$$

Last, for inputs of the form $\langle ?z(?x, ?y), \{?x\} \rangle$, the two cost components are computed as:

$$\left\langle \sum_{r \in N_R^{\mathcal{O}}} \left(\frac{|K[r]|}{|\mathsf{sucK}[r]|} \cdot d \cdot C_L + \frac{|P[r]|}{|\mathsf{sucP}[r]|} \cdot d \cdot C_E \right), \sum_{r \in N_R^{\mathcal{O}}} \left(\frac{|K[r]|}{|\mathsf{sucK}[r]|} + \frac{|P[r]|}{|\mathsf{sucP}[r]|} \cdot P_{IS} \right) \right\rangle$$

For concept (role) inclusion axiom templates of the form $c_{t_1} \sqsubseteq c_{t_2}$ ($r_{t_1} \sqsubseteq r_{t_2}$), where c_{t_1}, c_{t_2} concept terms (r_{t_1}, r_{t_2} role terms), we need look-ups in the computed concept (role) hierarchy in order to compute the answers (assuming that the concept (role) hierarchy is precomputed).

One can define similar cost functions for other types of axiom templates (complex axiom templates) by either using the available statistics or by relying on told information from the ontology. For this work, however, we just define a cost function based on the assumption that we iterate over all possible values of the respective variables and do one consistency check for each value. Hence, we define the following general cost function for these cases:

$$\langle |N| \cdot C_E, |N| \rangle,$$

where $N \in \{N_C^{\mathcal{O}}, N_R^{\mathcal{O}}, N_I^{\mathcal{O}}\}$ as appropriate for the variable that is tested. As discussed in Section 4.3.1, the dynamic function is based on the static one and is applied only to the above described cases for an empty set of bound variables.

Proposition 1. *Let q be a query over an ontology $\mathcal{O}$, s and d the static and dynamic cost functions defined in Sections 4.3.1 and 4.3.2. The ordering induced by s and d is a total order over the axiom templates of q.*

Proof. The cost functions s and d are defined for all kinds of axiom templates and return two real numbers to each possible input. Since, according to Definition 12, the orders $\preceq_s$ and $\preceq_d$ are based on the addition of the two real numbers, addition of reals yields again a real number, and since $\leq$ is a total order over the reals, we immediately get that $\preceq_s$ and $\preceq_d$ are total orders. □

It is obvious that the ordering of axiom templates does not affect soundness and completeness of a query evaluation algorithm.

4.4 Related Work

Techniques for ordering the atoms of a conjunctive instance query issued over an ontology have already been studied. As in databases, these techniques are based on the estimation of the costs of different join orderings and the selection of the ordering with the least cost. These costs are computed by preprocessing the queried ontology and extracting useful information. Well known examples of reasoners that use some sort of query ordering are Pellet [126] and Racer Pro [47] which are both tableau based reasoners.

The problem of finding good orderings for the templates of a query issued over an ontology has already been preliminarily studied [124, 84, 48]. Sirin et al. [124] explore query ordering for conjunctive instance queries and Kremen et al. [84] extend these query ordering techniques to queries expressed in SPARQL-DL [125], a language that allows to mix TBox and ABox queries that are similar to our queries with simple axiom templates only. Similarly to our work, Sirin et al. [124] as well as Kremen et al. [84] exploit reasoning techniques and information provided by reasoner models to create statistics about the cost and the result size of axiom template evaluations within execution plans. A difference is that they use cached models for cheaply finding obvious concept and role (non-)instances, whereas in our case we do not cache any model or model parts. Instead, we process the pre-model constructed for the initial ontology consistency check and extract the known and possible instances of concepts and roles from it. We subsequently use this information to create and update the query atom statistics. The use of the hypertableau algorithm often leads to less non-determinism which, informally, means that we can get on average more known and less possible instances for concept and role names from a hypertableau pre-model than from a pre-model constructed by a tableau algorithm. Moreover, Sirin et al. and Kremen et al. compare the costs of complete execution plans —after heuristically reducing the large number of possible complete plans — and choose the one that is most promising before the beginning of query execution. This is different from our cheap greedy algorithm that finds, at each iteration, the next most promising axiom template. Our experimental study shows that this is equally effective as the investigation of all possible execution orders.

In more detail, for the actual cost estimation, Kremen et al. use two functions for each axiom template, which take as input the queried ontology $\mathcal{O}$, the set of bound variables B and the atom at:

1. The function EC($\mathcal{O}$, B, at) estimates the cost of the ontology operation that is needed to evaluate the atom at based on the bound variables in B. The authors define six different ontology operation costs, namely, noSat, oneSat, instRetr, typeRetr, classify and realize for operations requiring no consistency check, one consistency check, a concept instance retrieval, a retrieval of types of an individual, classification and realization respectively. These costs are considered fixed values; they do not depend on the atom being evaluated.

2. The function EB($\mathcal{O}$, B, at) estimates the number of results that the evaluation of atom at will produce. To get such estimates the authors use information from the cached precompletion for conjunctive instance query atoms and told axioms to estimate the results of the remaining axiom templates.

We now show through an example how the costs of complete execution plans are determined in Kremen et al:

Example 9. *Let $\mathcal{O}$ be an ontology, q the following query:*

$$q = \{\mathsf{GraduateStudent}(?x), \mathsf{Woman}(?y), \mathsf{Professor}(?y), \mathsf{isAdvisedBy}(?x, ?y)\}$$

abbreviated as $\{\mathsf{GS}(?x), \mathsf{W}(?y), \mathsf{P}(?y), \mathsf{iAb}(?x, ?y)\}$ *and*

$$P = (\mathsf{GS}(?x), \mathsf{iAb}(?x, ?y), \mathsf{W}(?y), \mathsf{P}(?y))$$

an execution plan for q. The formula for the estimation of the cost of the execution plan is:

$$\begin{aligned} Cost =& \mathsf{EC}(\mathcal{O}, \emptyset, \mathsf{GS}(?x)) + \mathsf{EB}(\mathcal{O}, \emptyset, \mathsf{GS}(?x)) \cdot (\mathsf{EC}(\mathcal{O}, \{?x\}, \mathsf{iAb}(?x, ?y)) + \mathsf{EB}(\mathcal{O}, \{?x\}, \mathsf{iAb}(?x, ?y)) \\ & \cdot (\mathsf{EC}(\mathcal{O}, \{?x, ?y\}, \mathsf{W}(?y)) + \mathsf{EB}(\mathcal{O}, \{?x, ?y\}, \mathsf{W}(?y)) \cdot (\mathsf{EC}(\mathcal{O}, \{?x, ?y\}, \mathsf{P}(?y)) \\ & + \mathsf{EB}(\mathcal{O}, \{?x, ?y\}, \mathsf{P}(?y))))) = \mathsf{instRetr} + \mathsf{EB}(\mathcal{O}, \emptyset, \mathsf{GS}(?x)) \cdot (\mathsf{noSat} + \mathsf{EB}(\mathcal{O}, \{?x\}, \mathsf{iAb}(?x, ?y)) \\ & \cdot (\mathsf{oneSat} + \mathsf{EB}(\mathcal{O}, \{?x, ?y\}, \mathsf{W}(?y)) \cdot (\mathsf{oneSat} + \mathsf{EB}(\mathcal{O}, \{?x, ?y\}, \mathsf{P}(?y))))) \end{aligned}$$

For the case of conjunctive instance queries, as the one in Example 9, Sirin et al. perform the preprocessing described above and cache useful information for the estimation of the result output size EB of query concept and role atoms. Instead of checking all individuals of the queried ontology for determining concept and role atom sizes they take a random sample of individuals and check how many of the individuals in the sample are obvious (non-)instances without performing any consistency check and they then generalize the result to all individuals. According to an experimental evaluation of the authors [124], a 20 percent sample is a good percentage for accurate size estimation.

In the implementation of Pellet[2] the values for the ontology operation costs have been chosen rather randomly and do not accurately represent the actual costs. For example, the operation of checking whether an individual a is an instance of the concept A is always assigned a oneSat cost irrespective of whether a is a known instance of the concept A (which would require no consistency check) or an unknown instance of A (which would require one consistency check). In the same way, an atom that involves retrieving the instances of one concept is given a constant $\mathsf{instRetr}$ cost irrespective of which concept is involved (in reality retrieval queries for different concepts require a different number of consistency checks to be performed). In order to more accurately depict the real ontology operation cost we have 'quantified' the number of needed consistency checks by the number of possible instances of a concept or role and use this number for the cost computation. Moreover, we have defined different consistency check values depending on whether we refer to concept templates, role templates, equality atoms, sub-concept templates, sub-role templates or complex axiom templates. These values have been determined by taking the average of the running time of previously performed tests for each respective operation to better depict the cost of each operation.

Moreover, in Sirin et al. the results of joins between query atoms are taken into account through inaccurate average values in the cost computation. In order to find more accurate cost estimates in the presence of non-deterministic axioms, we have additionally used dynamic ordering and compared it to static ordering.

[2] `http://clarkparsia.com/pellet/`

According to the dynamic ordering algorithm, we order the templates while we evaluate the query. In each run of the algorithm, the costs of the next candidate atoms are determined by taking into account the solution mappings computed so far and the template with the least cost is added to the (partial) execution plan. In his work [83], Kremen has also presented a similar dynamic ordering algorithm. Since iterating over all individuals in each run is costly, we have further combined dynamic ordering with clustering techniques and have shown that these techniques lead to better performance particularly in ontologies that contain disjunctions and do not allow for purely deterministic reasoning.

Haarslev et al. [48] discuss by means of an informal example the ordering criteria they use to find efficient query execution plans for conjunctive instance queries in Racer Pro. In particular, they use traditional database cost based optimization techniques, which means that they take into account only the cardinality of concept and role atoms to decide about the most promising ordering exploiting techniques as the ones described in the beginning of the section. As previously discussed, this can be inadequate especially for ontologies with disjunctive information.

4.5 Extension of the Approach to Approximate Query Answering Systems

As it was stated in Section 4.2.1, the procedure we followed to extract sets of known concept and role instances can be seen as an approximate sound but incomplete instance retrieval algorithm for determining under-approximations of instances of concepts and roles, while the procedure for extracting sets of possible concept and role instances can be seen as an approximate complete but unsound instance retrieval algorithm for determining over-approximations of instances of concepts and roles. In fact, any approximate instance retrieval algorithm that satisfies the conditions of Definition 14 creates under- and over-approximations of instances of concept and roles. Such algorithm has the same interface as the algorithm we use for instance retrieval and hence can efficiently exploit the optimizations developed within our framework. More formally, the definition of an approximate instance retrieval algorithm is given below:

Definition 14 (**Approximate Instance Retrieval Algorithm**). *Let* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ *be an ontology and* at *a concept or role atom such that the concept or role is from the signature of* $\mathcal{O}$. *An* approximate instance retrieval algorithm $\mathsf{inst}(\mathcal{O}, \mathsf{at})$ *returns a pair of sets of (total) functions from* $\mathsf{Var}(\mathsf{at})$ *to individual names from* $\mathcal{O}$, $\langle K_t[\mathsf{at}], P_t[\mathsf{at}] \rangle$, *such that:*

1. *if* $\mu \in K_t[\mathsf{at}]$, *then* $\mathcal{O} \models \mu(\mathsf{at})$, *and*
2. *for each total function* μ *from* $\mathsf{Var}(\mathsf{at})$ *to individual names from* $\mathcal{O}$ *such that* $\mathcal{O} \models \mu(\mathsf{at})$, $\mu \in K_t[\mathsf{at}] \cup P_t[\mathsf{at}]$.

In Definition 14 we use $K_t[\mathsf{at}]$ and $P_t[\mathsf{at}]$ to denote the known and possible instances of the atom at. The relationship between the sets K_t and K, P_t and P (K and P were defined in Section 4.2.1) is given below, taking cases about the atom at into account that can appear in a conjunctive instance query (without loss of generality we assume that we have only concept and role atoms):

Algorithm 3 intersecQans$(\mathcal{O}, q)$

Input: $\mathcal{O}$: a $\mathcal{SROIQ}$ ontology
q: a conjunctive instance query
Output: $\langle K_t[q], P_t[q]\rangle$: $K_t[q], P_t[q]$ sets of known and possible answers for q

```
1: for at ∈ q do
2:    ⟨K_t[at], P_t[at]⟩ := inst(O, at)
3:    if K_t[q] and P_t[q] not initialised then
4:       ⟨K_t[q], P_t[q]⟩ := ⟨K_t[at], P_t[at]⟩
5:    else
6:       K_t[q] := K_t[q] ⋈ K_t[at]
7:       P_t[q] := (P_t[q] ⋈ P_t[at]) ∪ (K_t[q] ⋈ P_t[at]) ∪ (P_t[q] ⋈ K_t[at])
8:    end if
9: end for
10: return ⟨K_t[q], P_t[q]⟩
```

- if at is $A(?x)$, then $K_t[A(?x)] = K[A]$, $P_t[A(?x)] = P[A]$
- if at is $r(?x, ?y)$, then $K_t[r(?x, ?y)] = K[r]$, $P_t[r(?x, ?y)] = P[r]$
- if at is $r(a, ?y)$, then $K_t[r(a, ?y)] = \mathsf{sucK}[r, a]$, $P_t[r(a, ?y)] = \mathsf{sucP}[r, a]$
- if at is $r(?x, a)$, then $K_t[r(?x, a)] = \mathsf{preK}[r, a]$, $P_t[r(?x, a)] = \mathsf{preP}[r, a]$

In Algorithm 3 (intersecQans) we show how K_t and P_t can be extended to cover sets of query atoms and how the used instance retrieval algorithm and generally every approximate instance retrieval algorithm that satisfies the conditions of Definition 14 can be used to develop an approximate query answering algorithm that satisfies the conditions of Definition 15. We now define the conditions that an approximate query answering algorithm should satisfy.

Definition 15 (Approximate Query Answering Algorithm). *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ be an ontology, and q a conjunctive instance query. An* approximate query answering algorithm apprQA$(\mathcal{O}, q)$ *returns a pair of sets of (total) functions from* Var(q) *to individual names from $\mathcal{O}$, $\langle K_t[q], P_t[q]\rangle$, such that:*

1. *if $\mu \in K_t[q]$, then $\mu \in \mathsf{ans}(\mathcal{O}, q)$, and*
2. *for each total function μ from* Var(q) *to individual names from $\mathcal{O}$ such that $\mu \in \mathsf{ans}(\mathcal{O}, q)$, $\mu \in K_t[q] \cup P_t[q]$.*

Afterwards, we give an example that shows the intuition behind Algorithm 3 and how the algorithm is applied.

Example 10. *Let $\mathcal{O}$ be an ontology and $q = \{A(?x), r(?x, ?y), B(?y)\}$ a conjunctive instance query over $\mathcal{O}$. Suppose that for the known and possible instances of the query concepts and roles we have*

$$K[A] = \{a\} \qquad K[r] = \{\langle a, c\rangle\} \qquad K[B] = \{c, d\}$$
$$P[A] = \{b\} \qquad P[r] = \{\langle b, d\rangle, \langle b, e\rangle\} \qquad P[B] = \emptyset$$

From these sets, we can conclude that the mapping $\{?x \mapsto a, ?y \mapsto c\}$ *is a certain/known answer to* q*, since* $a \in K[A]$*,* $\langle a, c\rangle \in K[r]$ *and* $c \in K[B]$*. However, only the mapping* $\{?x \mapsto b, ?y \mapsto d\}$ *is a possible answer for* q *since* $b \in P[A]$*,* $\langle b, d\rangle \in P[r]$ *and* $d \in K[B]$*; the mapping* $\langle b, e\rangle$ *cannot be an answer for* q *since although* $\langle b, e\rangle \in P[r]$ *and* $b \in P[A]$*,* $e \notin K[B] \cup P[B]$*.*

We now show a run of Algorithm 3 on this example. If we take q *in the order given, we first initialize* $K_t[q]$ *to* $\{\{?x \mapsto a\}\}$ *and* $P_t[q]$ *to* $\{\{?x \mapsto b\}\}$*. During the next iterations, the (preliminary) set* $K_t[q]$ *is extended by performing a natural join with the known answers for the current atom* at*. The set* $P_t[q]$ *is extended by performing a natural join of it with both* $K_t[\mathsf{at}]$ *and* $P_t[\mathsf{at}]$ *and of* $K_t[q]$ *with* $P_t[\mathsf{at}]$ *in order to guarantee that all possible answers are preserved, i.e., we next process* $r(?x, ?y)$ *and obtain*

$$\begin{aligned} K_t[q] =& \{\{?x \mapsto a, ?y \mapsto c\}\}, \\ P_t[q] =& \{\{?x \mapsto b, ?y \mapsto d\}, \{?x \mapsto b, ?y \mapsto e\}\}. \end{aligned}$$

We finally process $B(?y)$ *and obtain*

$$\begin{aligned} K_t[q] &= \{\{?x \mapsto a, ?y \mapsto c\}\}, \\ P_t[q] &= \{\{?x \mapsto b, ?y \mapsto d\}\}, \end{aligned}$$

which is the output of Algorithm 3.

Lemma 1. *Algorithm* $\mathsf{intersecQans}$ *is an approximate query answering algorithm.*

Proof. We prove the lemma by induction on the length of the query q given as input to the algorithm $\mathsf{intersecQans}(\mathcal{O}, q)$.
Base case: For $q = \{\mathsf{at}\}$ the two conditions of the definition of approximate query answering algorithms are satisfied. Indeed,

1. if $\mu \in K_t[q]$, i.e., $\mu \in K_t[\mathsf{at}]$ then $\mathcal{O} \models \mu(\mathsf{at})$ (since $\mathsf{inst}(\mathcal{O}, \mathsf{at})$ is an approximate instance retrieval algorithm), which means that $\mathcal{O} \models \mu(q)$.

2. for each total function μ from $\mathsf{Var}(q)$ to individual names from $\mathcal{O}$ such that $\mu \in \mathsf{ans}(\mathcal{O}, q)$, i.e., $\mu \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}\})$, i.e., $\mathcal{O} \models \mu(\mathsf{at})$, it holds $\mu \in K_t[\mathsf{at}] \cup P_t[\mathsf{at}]$ (since $\mathsf{inst}(\mathcal{O}, \mathsf{at})$ is an approximate instance retrieval algorithm), i.e., $\mu \in K_t[q] \cup P_t[q]$.

Inductive Step: Let $q' = q \cup \{\mathsf{at}\}$ and let us assume that $\mathsf{intersecQans}(\mathcal{O}, q)$ is an approximate query answering algorithm (IH). We prove that $\mathsf{intersecQans}(\mathcal{O}, q')$ is also an approximate query answering algorithm by showing that the two conditions of Definition 15 hold for q'.

1. If $\mu \in K_t[q']$, this means that $\mu_{|\mathsf{Var}(q)} \in K_t[q]$ and $\mu_{|\mathsf{Var}(\mathsf{at})} \in K_t[\mathsf{at}]$ (it can easily be seen from the construction of $K_t[q']$ in $\mathsf{intersecQans}(\mathcal{O}, q')$). By induction hypothesis $\mu_{|\mathsf{Var}(q)} \in \mathsf{ans}(\mathcal{O}, q)$ and, since $\mathsf{inst}(\mathcal{O}, \mathsf{at})$ is an approximate instance retrieval algorithm, $\mathcal{O} \models \mu_{|\mathsf{Var}(\mathsf{at})}(\mathsf{at})$. Since $\mu = \mu_{|\mathsf{Var}(q)} \bowtie \mu_{|\mathsf{Var}(\mathsf{at})}$, it holds $\mu \in \mathsf{ans}(\mathcal{O}, q')$.

2. For each total function μ from $\mathsf{Var}(q')$ to individual names from $\mathcal{O}$ such that $\mu \in \mathsf{ans}(\mathcal{O}, q')$, it holds $\mu_{|\mathsf{Var}(q)} \in \mathsf{ans}(\mathcal{O}, q)$ and $\mathcal{O} \models \mu_{|\mathsf{Var}(\mathsf{at})}(\mathsf{at})$. By induction

hypothesis, $\mu_{|\mathsf{Var}(q)} \in K_t[q] \cup P_t[q]$ and $\mu_{|\mathsf{Var}(\mathsf{at})} \in K_t[\mathsf{at}] \cup P_t[\mathsf{at}]$, since $\mathsf{inst}(\mathcal{O}, \mathsf{at})$ is an approximate instance retrieval algorithm. It holds $\mu = \mu_{|\mathsf{Var}(q)} \bowtie \mu_{|\mathsf{Var}(\mathsf{at})}$. We distinguish between the following cases (these can easily be seen from the construction of $P_t[q']$ in $\mathsf{intersecQans}(\mathcal{O}, q')$):

- if $\mu_{|\mathsf{Var}(q)} \in K_t[q]$ and $\mu_{|\mathsf{Var}(\mathsf{at})} \in K_t[\mathsf{at}]$, then $\mu \in K_t[q']$
- if $\mu_{|\mathsf{Var}(q)} \in K_t[q]$ and $\mu_{|\mathsf{Var}(\mathsf{at})} \in P_t[\mathsf{at}]$, then $\mu \in P_t[q']$
- if $\mu_{|\mathsf{Var}(q)} \in P_t[q]$ and $\mu_{|\mathsf{Var}(\mathsf{at})} \in K_t[\mathsf{at}]$, then $\mu \in P_t[q']$
- if $\mu_{|\mathsf{Var}(q)} \in P_t[q]$ and $\mu_{|\mathsf{Var}(\mathsf{at})} \in P_t[\mathsf{at}]$, then $\mu \in P_t[q']$

We see from the above that in any case $\mu \in K_t[q'] \cup P_t[q']$.

□

We now briefly discuss the relation between our algorithm (if we treat the sets of known and possible concept and role instances as under- and over-approximations of instances of query concepts and roles) and approximate query answering techniques proposed in the literature [102, 143]. The works of Pan et al. [102] and Zhou et al. [143] described in Section 3.2.4 are based on an under-approximation and an over-approximation set for a query q, which can be seen as our $K_t[q]$ and $P_t[q]$ sets. The approach by Pan et al. computes the materialization w.r.t. concept and role name assertions of the OWL DL ontology in the process of transforming the initial queried OWL DL ontology $\mathcal{O}$ to a DL-Lite ontology $\mathcal{O}_L$. In other words, every concept and role name assertion containing terms from the signature of $\mathcal{O}$ is created and checked whether it is entailed by $\mathcal{O}$ before the beginning of the query answering procedure, which can be expensive. This is different from the approach we use, in which we do not materialize every concept name and role name assertion beforehand; we do it only on demand when a query asks for it. Zhou et al. creates under- and over-approximations by transforming the queried ontology to two OWL RL ontologies, which are then used to find under- and over-approximations of answers to conjunctive queries using an OWL RL engine as described in Section 3.2.4.

4.6 Discussion

In this section we widen the applicability of the presented cost-based query ordering techniques for the case of conjunctive instance queries. As it has been stated before, the presented cost functions can be used with any (hyper)tableau reasoning system, since any reasoner holds information about deterministically and non-deterministically derived assertions. The presented query ordering techniques can also be used when optimizations such as caching, absorption, pseudo model merging, binary instance retrieval or lazy unfolding (discussed in Section 4.2.1) are employed. The cost functions might, however, require some adaptation to take the reduction in the required number of consistency checks into account. For example, the cost functions need to take into account cached pre-completion information, i.e., the number of obvious concept and role instances need to be estimated and used in the computation of the cost functions. The possible instances can be determined by the assertions that are derived non-deterministically as discussed in Section 4.2. The use of absorption techniques in tableau algorithms can reduce the number of possible

instances. When pseudo-model techniques are used, the set of possible instances can be reduced by determining obvious non-instances among possible instances, something which needs to be depicted in the cost functions. The cost functions should also take into account the reduction in the number of possible instances when the binary instance retrieval optimization is used, something which may be more difficult to determine beforehand.

The presented query ordering cost functions are general and can be used by any of the query answering algorithms presented in Chapter 3. In deterministic ontologies, like OWL 2 RL and OWL 2 QL, the cost functions are based only on the cardinality of the sets of known instances of concept and role names, since, in such ontologies, the sets of possible instances for all concepts and roles are empty. Note that in OWL 2 RL and OWL 2 QL traditional query ordering techniques from databases and triple stores can also be applied. As we have discussed in Section 3.2.1, in OWL 2 RL ontologies the complete materialization is computed before the beginning of the query answering procedure. This materialization set, which is used to find sound and complete answers to conjunctive queries, can be seen as an (extended) database and hence traditional query ordering techniques from databases and triple stores can be used to find good query orders. Similarly, in OWL 2 QL ontologies, the rewritten conjunctive queries are evaluated over databases or triple stores and hence traditional query ordering techniques can be used.

Chapter 5

Optimizations for Complex Axiom Templates

In this chapter we describe optimizations we have developed for complex axiom templates.

5.1 Axiom Template Rewriting

Some costly to evaluate axiom templates can be rewritten into axiom templates that can be evaluated more efficiently and yield an equivalent result. Before we go on to describe the axiom template rewriting technique, we define what a concept template is, which is useful throughout the section.

Definition 16 (**Concept Template**). *Let $\mathcal{S}_q = (N_C, N_R, N_R, V_C, V_R, V_I)$ be a query signature w.r.t. a signature $\mathcal{S} = (N_C, N_R, N_I)$. A concept template over $\mathcal{S}_q$ is a $\mathcal{SROIQ}$ concept over $\mathcal{S}$, where one can also use concept variables from V_C in place of concept names, role variables from V_R in place of role names and individual variables from V_I in place of individual names.*

Definition 17 (**Rewriting**). *Let* at *be an axiom template over $\mathcal{S}_q$, $t, t_1, \ldots t_n$ individuals or individual variables from $\mathcal{S}_q$, and $C, C_1, \ldots, C_n$ concept templates over $\mathcal{S}_q$. The function* rewrite *takes an axiom template and returns a set of axiom templates as follows:*

- *if* $\mathsf{at} = (C_1 \sqcap \ldots \sqcap C_n)(t)$, *then* $\mathsf{rewrite}(\mathsf{at}) = \{C_1(t), \ldots, C_n(t)\}$;
- *if* $\mathsf{at} = C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$, *then* $\mathsf{rewrite}(\mathsf{at}) = \{C \sqsubseteq C_1, \ldots, C \sqsubseteq C_n\}$;
- *if* $\mathsf{at} = C_1 \sqcup \ldots \sqcup C_n \sqsubseteq C$, *then* $\mathsf{rewrite}(\mathsf{at}) = \{C_1 \sqsubseteq C, \ldots, C_n \sqsubseteq C\}$.
- *if* $\mathsf{at} = t_1 \approx \ldots \approx t_n$, *then* $\mathsf{rewrite}(\mathsf{at}) = \{t_1 \approx t_2, t_2 \approx t_3, \ldots, t_{n-1} \approx t_n\}$.

To understand the intuition behind such transformation, we consider a query with only the axiom template: $?x \sqsubseteq \exists r.?y \sqcap A$. Its evaluation requires a quadratic number of consistency checks in the number of concepts (since $?x$ and $?y$ are concept variables). The rewriting yields: $?x \sqsubseteq A$ and $?x \sqsubseteq \exists r.?y$. The first axiom template is now evaluated with a cheap cache look-up (assuming that the concept hierarchy has been precomputed). For the second one, we only have to check the usually few resulting bindings for $?x$ combined with all other concept names for $?y$.

Note that Description Logics typically do not support n-ary equality axioms $t_1 \approx \ldots \approx t_n$, but only binary ones, whereas in OWL, one can typically also write n-ary equality axioms. Since our cost functions are only defined for binary equality axioms, we equivalently rewrite an n-ary one into several binary ones. One could even further optimize the evaluation of such atoms by just evaluating one binary equality axiom template and by then propagating the binding for the found equivalent individuals to the other equality axioms. This is valid since equality is a congruence relation.

It can be easily proved that for any ontology $\mathcal{O}$, $\mathsf{ans}(\mathcal{O}, \{\mathsf{at}\}) = \mathsf{ans}(\mathcal{O}, \mathsf{rewrite}(\mathsf{at}))$.

5.2 Concept and Role Hierarchy Exploitation

The number of consistency checks required to evaluate a query can be further reduced by taking the concept and role hierarchies into account. Once the concepts and roles are classified (this can ideally be done before a system accepts queries), the hierarchies are stored in the reasoner's internal structures. We further use the hierarchies to prune the search space of solutions in the evaluation of certain axiom templates. We illustrate the intuition with the example $\mathsf{Infection} \sqsubseteq \exists \mathsf{hasCausalLinkTo}.?x$. If A is not a solution and $B \sqsubseteq A$ holds, then B is also not a solution. Thus, when searching for solutions for $?x$, we choose the next binding to test by traversing the concept hierarchy top-down. When we find a non-solution A, the subtree rooted in A of the concept hierarchy can safely be pruned. Queries over ontologies with a large number of concepts and a deep concept hierarchy can, therefore, gain the maximum advantage from this optimization. We employ similar optimizations using the role hierarchies.

In the example above, we can prune the sub-concepts of A because $?x$ has positive polarity in the axiom template $\mathsf{Infection} \sqsubseteq \exists \mathsf{hasCausalLinkTo}.?x.$, i.e., $?x$ occurs positively on the right hand side of the axiom template. In case a variable $?x$ has negative polarity in an axiom template of the form $C_1 \sqsubseteq C_2$, i.e., $?x$ occurs directly or indirectly under a negation on the right hand side of the axiom template or positively on the left-hand side of the axiom template, one can, instead, prune the super-concepts.

We now run an example that shows in more detail how the concept-role polarity optimization is used for pruning query answers in the process of finding answers to a complex query. Let us assume that we have an ontology $\mathcal{O}$ containing a TBox $\mathcal{T} = \{s \sqsubseteq r, B \sqsubseteq \exists r.A\}$ and an empty ABox with $N_C^{\mathcal{O}} = \{\top, A, B, \bot\}$ and $N_R^{\mathcal{O}} = \{\top_r, r, s, \bot_r\}$ and that we have classified the TBox and have computed the concept and role hierarchies shown in Figure 5.1. And let us assume that we want to answer the query

$$q = \{B \sqsubseteq \exists ?y.?x\}$$

Both variables $?y$ and $?x$ have positive polarity in the axiom template of the query and hence we start by checking the mapping:

$$\mu\colon \quad \mu(?x) = \top,\ \mu(?y) = \top_r$$

This mapping is an answer for q over $\mathcal{O}$, and hence we create new mappings for each of the direct sub-concepts of the mapping for $?x$, i.e., $\top$, and for each of the direct

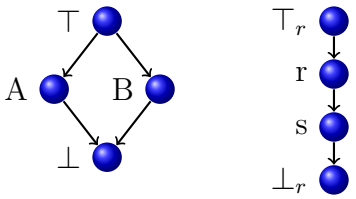

Figure 5.1: *Concept and role hierarchies*

sub-roles of the mapping for $?y$, i.e., $\top_r$. The new mappings are:

$$\begin{array}{ll} \mu_1\text{:} & \mu_1(?x) = A,\ \mu_1(?y) = \top_r \\ \mu_2\text{:} & \mu_2(?x) = B,\ \mu_2(?y) = \top_r \\ \mu_3\text{:} & \mu_3(?x) = \top, \mu_3(?y) = r \end{array}$$

Since all these three mappings are answers we expand them further by taking direct sub-concepts for the mappings for $?x$ and direct sub-roles for the mappings for $?y$. Hence, we have the following mappings

- for μ_1:

$$\begin{array}{ll} \mu_4\text{:} & \mu_4(?x) = A,\ \mu_4(?y) = r \\ \mu_5\text{:} & \mu_5(?x) = \bot,\ \mu_5(?y) = \top_r \end{array}$$

- for μ_2:

$$\begin{array}{ll} \mu_6\text{:} & \mu_6(?x) = B,\ \mu_6(?y) = r \\ \mu_7\text{:} & \mu_7(?x) = \bot,\ \mu_7(?y) = \top_r \end{array}$$

- for μ_3:

$$\begin{array}{ll} \mu_8\text{:} & \mu_8(?x) = A,\ \mu_8(?y) = r \\ \mu_9\text{:} & \mu_9(?x) = B,\ \mu_9(?y) = r \\ \mu_{10}\text{:} & \mu_{10}(?x) = \top,\ \mu_{10}(?y) = s \end{array}$$

From these mappings only μ_4 (and μ_8, which is the same) is an answer and hence we take the following mappings, which in the end are not answers:

$$\begin{array}{ll} \mu_{11}\text{:} & \mu_{11}(?x) = \bot,\ \mu_{11}(?y) = r \\ \mu_{12}\text{:} & \mu_{12}(?x) = A,\ \mu_{12}(?y) = s \end{array}$$

From the above example, we see that we have overall tested 13 mappings, whereas if we had tried every combination of variable mappings, we would have tested $4{\times}4 = 16$ mappings. It is obvious that the proposed hierarchy traversal can lead to the creation of the same mappings more than once. This means that by caching mappings that have already been tested we can further reduce the mappings that need to be tested. In the above example, the mappings μ_4 and μ_8 are the same as are the mappings μ_5 and μ_7 and the mappings μ_6 and μ_9. If we perform the above described optimizations, the number of tests are reduced to 10.

We next specify more precisely the polarity of a concept variable in a concept template or in an axiom template.

Definition 18 (**Concept Polarity**). *Let* $?x \in V_C$ *be a concept variable and* C, C_1, C_2, D *concept templates,* r *a role, and* $n \in \mathbb{N}_0$. *We define the* polarity *of* $?x$ *in* C *as follows:* $?x$ occurs positively *in* $?x$. *Furthermore,* $?x$ occurs positively (negatively)

- *in* $\neg D$ *if* $?x$ *occurs negatively (positively) in* D,
- *in* $C_1 \sqcap C_2$ *or* $C_1 \sqcup C_2$ *if* $?x$ *occurs positively (negatively) in* C_1 *or* C_2,
- *in* $\exists r.D$, $\forall r.D$, *or* $\geqslant n\ r.D$ *if* $?x$ *occurs positively (negatively) in* D,
- *in* $\leqslant n\ r.D$ *if* $?x$ *occurs negatively (positively) in* D
- *in* $= n\ r.D$ *if* $?x$ *occurs in* D.

We further say that $?x$ *occurs positively* (*negatively*) *in* $C_1 \sqsubseteq C_2$ *if* $?x$ *occurs negatively* (*positively*) *in* C_1 *or positively* (*negatively*) *in* C_2. *We further define a partial function* pol_c *that maps a concept variable* $?x$ *and a concept template* C (*axiom template of the form* $C_1 \sqsubseteq C_2$) *to* pos *if* $?x$ *occurs* only positively *in* C ($C_1 \sqsubseteq C_2$) *and to* neg *if* $?x$ *occurs* only negatively *in* C ($C_1 \sqsubseteq C_2$).

Note that no matter whether $?x$ occurs positively or negatively in a concept template D, in any concept template C of the form $= n\ r.D$, $?x$ occurs positively as well as negatively. This is due to the fact that C is equivalent to the concept template $\leqslant n\ r.D \sqcap \geqslant n\ r.D$ in which $?x$ occurs positively and negatively. Since the function pol_c is not defined for variables that appear both positively and negatively, the concept hierarchy cannot be exploited in this case. For example, consider the concept template $\neg ?x \sqcup \exists r.?x$, (axiom template $?x \sqsubseteq \exists r.?x$), where $?x$ appears both positively and negatively. It is obvious that if $\mathcal{O} \models A \sqsubseteq \exists r.A$, where $\mathcal{O}$ is an ontology and either $\mathcal{O} \models A \sqsubseteq B$ or $\mathcal{O} \models B \sqsubseteq A$, it does not hold that $\mathcal{O} \models B \sqsubseteq \exists r.B$.

Before proving the correctness of the proposed optimization, we first show the relationship between entailment and concept membership, which is used in the subsequent proofs.

Lemma 2. *Let* at *be an axiom template over an ontology* $\mathcal{O}$ *of the form* $C_1 \sqsubseteq C_2$ *and* μ *a mapping for* at *over* $\mathcal{O}$. *It holds that* $\mathcal{O} \not\models \mu(C_1 \sqsubseteq C_2)$ *iff there exists an interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *and an element* $\delta \in \Delta^{\mathcal{I}}$ *such that* $\mathcal{I} \models \mathcal{O}$ *and* $\delta \notin \mu(\neg C_1 \sqcup C_2)^{\mathcal{I}}$.

Proof. $\mathcal{O} \not\models \mu(C_1 \sqsubseteq C_2)$ holds iff there exists an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and an element $\delta \in \Delta^{\mathcal{I}}$ such that $\mathcal{I} \models \mathcal{O}$ and $\delta \in \mu(C_1)^{\mathcal{I}}$ and $\delta \notin \mu(C_2)^{\mathcal{I}}$, which holds iff $\delta \in \mu(C_1)^{\mathcal{I}}$ and $\delta \in \mu(\neg C_2)^{\mathcal{I}}$, which is equivalent to $\delta \in \mu(C_1 \sqcap \neg C_2)^{\mathcal{I}}$, which is equivalent to $\delta \in \mu(\neg(\neg C_1 \sqcup C_2))^{\mathcal{I}}$, which holds iff $\delta \notin \mu(\neg C_1 \sqcup C_2)^{\mathcal{I}}$. □

The following theorem holds for every axiom template of the form $C_1 \sqsubseteq C_2$. Note that we assume here that concept assertion templates of the form $C(a)$ are expressed as the equivalent axiom templates $\{a\} \sqsubseteq C$. We use $C_{\mu(?x)=A}$, where A is a concept name, to denote the concept obtained by applying the extension of μ that also maps $?x$ to A.

Theorem 1. *Let* $\mathcal{O}$ *be an ontology,* A, B *concept names such that* $\mathcal{O} \models A \sqsubseteq B$, C_1, C_2 *concept templates,* $C_1 \sqsubseteq C_2$ *an axiom template,* $C = \neg C_1 \sqcup C_2$, $?x \in V_C$ *a concept variable occurring in* C *and* μ *a mapping that covers all variables of* C *apart from* $?x$.

1. *For* $\mathsf{pol}_c(?x, C) = \mathsf{pos}$ *the following holds:*
   if $\mathcal{O} \not\models (C_1 \sqsubseteq C_2)_{\mu(?x)=B}$*, then* $\mathcal{O} \not\models (C_1 \sqsubseteq C_2)_{\mu(?x)=A}$
2. *For* $\mathsf{pol}_c(?x, C) = \mathsf{neg}$ *the following holds:*
   if $\mathcal{O} \not\models (C_1 \sqsubseteq C_2)_{\mu(?x)=A}$*, then* $\mathcal{O} \not\models (C_1 \sqsubseteq C_2)_{\mu(?x)=B}$

Proof. Due to Lemma 2, it suffices to show for every model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{O}$ and element $\delta \in \Delta^{\mathcal{I}}$ the following (which is formalized in contrapositive form):

1. For $\mathsf{pol}_c(?x, C) = \mathsf{pos}$ it holds that if $\delta \in (C_{\mu(?x)=A})^{\mathcal{I}}$, then $\delta \in (C_{\mu(?x)=B})^{\mathcal{I}}$.
2. For $\mathsf{pol}_c(?x, C) = \mathsf{neg}$ it holds that if $\delta \in (C_{\mu(?x)=B})^{\mathcal{I}}$, then $\delta \in (C_{\mu(?x)=A})^{\mathcal{I}}$.

We prove the claim by induction on the structure of the concept template C:

- For $C = ?x$, $?x$ occurs positively in C. Now, if $\delta \in (?x_{\mu(?x)=A})^{\mathcal{I}}$, that is $\delta \in A^{\mathcal{I}}$, it is easy to see that $\delta \in B^{\mathcal{I}}$ since $\mathcal{O} \models A \sqsubseteq B$ by assumption. Hence, $\delta \in (?x_{\mu(?x)=B})^{\mathcal{I}}$.
- For $C = \neg D$ and $\mathsf{pol}_c(?x, C) = \mathsf{pos}$, if $\delta \in (\neg D_{\mu(?x)=A})^{\mathcal{I}}$, we have to show that $\delta \in (\neg D_{\mu(?x)=B})^{\mathcal{I}}$. Note that $\mathsf{pol}_c(?x, D) = \mathsf{neg}$. In contrary to what is to be shown, assume that $\delta \in (D_{\mu(?x)=B})^{\mathcal{I}}$. Since $\mathcal{O} \models A \sqsubseteq B$ and by induction hypothesis $\delta \in (D_{\mu(?x)=A})^{\mathcal{I}}$ which is a contradiction. The proof is analogous for $\mathsf{pol}_c(?x, C) = \mathsf{neg}$.
- For $C = C_1 \sqcap C_2$ and $\mathsf{pol}_c(?x, C) = \mathsf{pos}$, if $\delta \in ((C_1 \sqcap C_2)_{\mu(?x)=A})^{\mathcal{I}}$, then $\delta \in (C_{1\mu(?x)=A})^{\mathcal{I}}$ and $\delta \in (C_{2\mu(?x)=A})^{\mathcal{I}}$. Since $\mathcal{O} \models A \sqsubseteq B$ and by induction hypothesis, $\delta \in (C_{1\mu(?x)=B})^{\mathcal{I}}$ and $\delta \in (C_{2\mu(?x)=B})^{\mathcal{I}}$. Thus, $\delta \in ((C_1 \sqcap C_2)_{\mu(?x)=B})^{\mathcal{I}}$. The proof is analogous for $\mathsf{pol}_c(?x, C) = \mathsf{neg}$.
- The proof for $C_1 \sqcup C_2$ is analogous to the one for $C_1 \sqcap C_2$.
- For $C = \exists r.D$ and $\mathsf{pol}_c(?x, C) = \mathsf{pos}$, if $\delta \in ((\exists r.D)_{\mu(?x)=A})^{\mathcal{I}}$, then δ has at least one r-successor, say δ', that is an instance of $D_{\mu(?x)=A}$. Since $\mathcal{O} \models A \sqsubseteq B$ and by induction hypothesis, $\delta' \in D_{\mu(?x)=B}$. Hence, $\delta \in (\exists r.(D_{\mu(?x)=B}))^{\mathcal{I}} = ((\exists r.D)_{\mu(?x)=B})^{\mathcal{I}}$. The proof is analogous for $\mathsf{pol}_c(?x, C) = \mathsf{neg}$.
- For $C = \forall r.D$ and $\mathsf{pol}_c(?x, C) = \mathsf{pos}$, if $\delta \in ((\forall r.D)_{\mu(?x)=A})^{\mathcal{I}}$, then $\delta \in (\forall r.(D)_{\mu(?x)=A})^{\mathcal{I}}$ and each r-successor of δ is an instance of $D_{\mu(?x)=A}$. Since $\mathcal{O} \models A \sqsubseteq B$ and by induction hypothesis, these r-successors are also instances of $D_{\mu(?x)=B}$. Hence, $\delta \in (\forall r.(D_{\mu(?x)=B}))^{\mathcal{I}} = ((\forall r.D)_{\mu(?x)=B})^{\mathcal{I}}$. The proof is analogous for $\mathsf{pol}_c(?x, C) = \mathsf{neg}$.
- For $C = \geqslant n\ r.D$ and $\mathsf{pol}_c(?x, C) = \mathsf{pos}$, if $\delta \in ((\geqslant n\ r.D)_{\mu(?x)=A})^{\mathcal{I}}$, then δ has at least n distinct r-successors which are instances of $D_{\mu(?x)=A}$. Since $\mathcal{O} \models A \sqsubseteq B$ and by induction hypothesis, these successors are instances of $D_{\mu(?x)=B}$. Hence, δ has at least n distinct r-successors that are instances of $D_{\mu(?x)=B}$ and, therefore, $\delta \in (\geqslant n\ r.(D)_{\mu(?x)=B})^{\mathcal{I}} = ((\geqslant n\ r.D)_{\mu(?x)=B})^{\mathcal{I}}$. The proof is analogous for $\mathsf{pol}_c(?x, C) = \mathsf{neg}$.
- For $C = \leqslant n\ r.D$ and $\mathsf{pol}_c(?x, C) = \mathsf{pos}$, if $\delta \in ((\leqslant n\ r.D)_{\mu(?x)=A})^{\mathcal{I}}$, we have to show that $\delta \in ((\leqslant n\ r.D)_{\mu(?x)=B})^{\mathcal{I}}$. Note that $\mathsf{pol}_c(?x, D) = \mathsf{neg}$. In contrary to what is to be shown, assume that $\delta \in (\neg(\leqslant n\ r.D)_{\mu(?x)=B})^{\mathcal{I}}$,

i.e., $\delta \in ((\geqslant\ n+1\ r.D)_{\mu(?x)=B})^{\mathcal{I}}$. Hence, δ has at least $n+1$ distinct r-successors which are instances of $D_{\mu(?x)=B}$. Since $\mathsf{pol}_c(?x, D) = \mathsf{neg}$ and by induction hypothesis, these $D_{\mu(?x)=B}$ instances are also $D_{\mu(?x)=A}$ instances and $\delta \in (\geqslant n{+}1\ r.(D)_{\mu(?x)=A})^{\mathcal{I}} = ((\geqslant n{+}1\ r.D)_{\mu(?x)=A})^{\mathcal{I}}$, which is a contradiction. The proof is analogous for $\mathsf{pol}_c(?x, C) = \mathsf{neg}$.

- For $C = (= n\ r.D)$, the polarity of $?x$ in C is always positive and negative, so $\mathsf{pol}_c(?x, C)$ is undefined and the case cannot occur.

□

We now extend this optimization to the case of role variables and we first define the polarity of a role variable in a concept template or in an axiom template.

Definition 19 (Role Polarity). *Let $?x \in V_R$ be a role variable, C, C_1, C_2, D concept templates, r a role, and $n \in \mathbb{N}_0$. We define the* polarity *of $?x$ in C as follows:* $?x$ occurs positively *in $\exists ?x.D$, $\exists ?x^-.D$, $\geqslant n\ ?x.D$, $\geqslant n\ ?x^-.D$, $= n\ ?x.D$, and $= n\ ?x^-.D$;* $?x$ occurs negatively *in $\forall ?x.D$, $\forall ?x^-.D$, $\leqslant n\ ?x.D$, $\leqslant n\ ?x^-.D$, $= n\ ?x.D$, and $= n\ ?x^-.D$. Furthermore, $?x$ occurs positively (negatively)*

- *in $\neg D$ if $?x$ occurs negatively (positively) in D,*
- *in $C_1 \sqcap C_2$ or $C_1 \sqcup C_2$ if $?x$ occurs positively (negatively) in C_1 or C_2,*
- *in $\exists r.D$, $\exists ?x.D$, $\exists ?x^-.D$, $\geqslant n\ r.D$, $\geqslant n\ ?x.D$, $\geqslant n\ ?x^-.D$, $\forall r.D$, $\forall ?x.D$, or $\forall ?x^-.D$ if $?x$ occurs positively (negatively) in D,*
- *in $\leqslant n\ r.D$, $\leqslant n\ ?x.D$, or $\leqslant n\ ?x^-.D$ if $?x$ occurs negatively (positively) in D,*
- *in $= n\ r.D$ if $?x$ occurs in D.*

We further say that $?x$ occurs positively (negatively) in $C_1 \sqsubseteq C_2$ if $?x$ occurs negatively (positively) in C_1 or positively (negatively) in C_2. We define a partial function pol_r that maps a role variable $?x$ and a concept template C (axiom template of the form $C_1 \sqsubseteq C_2$) to pos *if $?x$ occurs* only positively *in C ($C_1 \sqsubseteq C_2$) and to* neg *if $?x$ occurs* only negatively *in C ($C_1 \sqsubseteq C_2$).*

We now show, that the hierarchy optimization is also applicable to role variables, provided they occur only positively or only negatively.

Theorem 2. *Let $\mathcal{O}$ be an ontology, r, s role names such that $\mathcal{O} \models r \sqsubseteq s$, C_1, C_2 concept templates, $C_1 \sqsubseteq C_2$ an axiom template, $C = \neg C_1 \sqcup C_2$, $?x \in V_R$ a role variable occurring in C and μ a mapping that covers all variables of C apart from $?x$.*

1. *For $\mathsf{pol}_r(?x, C) = \mathsf{pos}$ the following holds:*
   if $\mathcal{O} \not\models (C_1 \sqsubseteq C_2)_{\mu(?x)=s}$, then $\mathcal{O} \not\models (C_1 \sqsubseteq C_2)_{\mu(?x)=r}$.
2. *For $\mathsf{pol}_r(?x, C) = \mathsf{neg}$ the following holds:*
   if $\mathcal{O} \not\models (C_1 \sqsubseteq C_2)_{\mu(?x)=r}$, then $\mathcal{O} \not\models (C_1 \sqsubseteq C_2)_{\mu(?x)=s}$.

Proof. Due to Lemma 2, it suffices to show for every model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{O}$ and element $\delta \in \Delta^{\mathcal{I}}$ the following (which is formalized in contrapositive form):

1. For $\mathsf{pol}_r(?x, C) = \mathsf{pos}$ it holds that if $\delta \in (C_{\mu(?x)=r})^{\mathcal{I}}$, then $\delta \in (C_{\mu(?x)=s})^{\mathcal{I}}$.

2. For $\mathsf{pol}_r(?x, C) = \mathsf{neg}$ it holds that if $\delta \in (C_{\mu(?x)=s})^{\mathcal{I}}$, then $\delta \in (C_{\mu(?x)=r})^{\mathcal{I}}$.

We prove the claim by induction on the structure of the concept template C:

- For $C = \exists ?x.D$, where D is a concept template that does not contain $?x$. We have $\mathsf{pol}_r(?x, C) = \mathsf{pos}$. Assume, $\delta \in ((\exists ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$, that is, $\delta \in (\exists r.\mu(D))^{\mathcal{I}}$. Then there is some $\delta' \in \Delta^{\mathcal{I}}$ such that $\langle \delta, \delta' \rangle \in r^{\mathcal{I}}$ and $\delta' \in \mu(D)^{\mathcal{I}}$. Since $\mathcal{O} \models r \sqsubseteq s$, we also have $\langle \delta, \delta' \rangle \in s^{\mathcal{I}}$ and, therefore, $\delta \in (\exists s.\mu(D))^{\mathcal{I}} = ((\exists ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$.

- For $C = \forall ?x.D$, where D is a concept template that does not contain $?x$. We have $\mathsf{pol}_r(?x, C) = \mathsf{neg}$. If $\delta \in ((\forall ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$, we have to show that $\delta \in ((\forall ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$. In contrary to what is to be shown, assume that $\delta \in (\neg(\forall ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$, i.e., $\delta \in (\exists r.\mu(\neg D))^{\mathcal{I}}$. Hence, there is some $\delta' \in \Delta^{\mathcal{I}}$ such that $\langle \delta, \delta' \rangle \in r^{\mathcal{I}}$ and $\delta' \in \mu(\neg D)^{\mathcal{I}}$. Since $\mathcal{O} \models r \sqsubseteq s$, we also have $\langle \delta, \delta' \rangle \in s^{\mathcal{I}}$ and, therefore, $\delta \notin (\forall s.\mu(D))^{\mathcal{I}} = ((\forall ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$, which is a contradiction.

- For $C = \geqslant n\ ?x.D$ where D is a concept template that does not contain $?x$. We have $\mathsf{pol}_r(?x, C) = \mathsf{pos}$. Assume, $\delta \in ((\geqslant n\ ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$, that is $\delta \in (\geqslant n\ r.\mu(D))^{\mathcal{I}}$ and δ has at least n distinct r-successors which are instances of $\mu(D)$. Since $\mathcal{O} \models r \sqsubseteq s$ these r-successors are also s-successors of δ and, therefore, $\delta \in (\geqslant n\ s.\mu(D))^{\mathcal{I}} = ((\geqslant n\ ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$.

- For $C = \leqslant n\ ?x.D$ where C is a concept template that does not contain $?x$. We have $\mathsf{pol}_r(?x, C) = \mathsf{neg}$. If $\delta \in ((\leqslant n\ ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$, we have to show that $\delta \in ((\leqslant n\ ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$. In contrary to what is to be shown, assume that $\delta \in (\neg(\leqslant n\ ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$, i.e., $\delta \in (\geqslant n+1\ r.\mu(D))^{\mathcal{I}}$. Hence, δ has at least $n+1$ distinct r-successors, which are instances of $\mu(D)$. Since $\mathcal{O} \models r \sqsubseteq s$, these r-successors are also s-successors and $\delta \in ((\geqslant n+1\ s.\mu(D)))^{\mathcal{I}} = ((\geqslant n+1\ ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$, which is a contradiction.

- For $C = C_1 \sqcap C_2$ and $\mathsf{pol}_r(?x, C) = \mathsf{pos}$, if $\delta \in ((C_1 \sqcap C_2)_{\mu(?x)=r})^{\mathcal{I}}$, then $\delta \in (C_{1\mu(?x)=r})^{\mathcal{I}}$ and $\delta \in (C_{2\mu(?x)=r})^{\mathcal{I}}$. Since $\mathcal{O} \models r \sqsubseteq s$ and by induction hypothesis, $\delta \in (C_{1\mu(?x)=s})^{\mathcal{I}}$ and $\delta \in (C_{2\mu(?x)=s})^{\mathcal{I}}$. Thus, $\delta \in ((C_1 \sqcap C_2)_{\mu(?x)=s})^{\mathcal{I}}$. The proof is analogous for $\mathsf{pol}_r(?x, C) = \mathsf{neg}$.

- The proof for $C_1 \sqcup C_2$ is analogous to the one for $C_1 \sqcap C_2$.

- For $C = \neg D$ and $\mathsf{pol}_r(?x, C) = \mathsf{pos}$, if $\delta \in (\neg D_{\mu(?x)=r})^{\mathcal{I}}$, we have to show that $\delta \in (\neg D_{\mu(?x)=s})^{\mathcal{I}}$. Note that $\mathsf{pol}_r(?x, D) = \mathsf{neg}$. In contrary to what is to be shown, assume that $\delta \in (D_{\mu(?x)=s})^{\mathcal{I}}$. Since $\mathcal{O} \models r \sqsubseteq s$ and by induction hypothesis $\delta \in (D_{\mu(?x)=r})^{\mathcal{I}}$ which is a contradiction. The proof is analogous for $\mathsf{pol}_r(?x, C) = \mathsf{neg}$.

- For $C = \exists p.D$ and $\mathsf{pol}_r(?x, C) = \mathsf{pos}$, we also have $\mathsf{pol}_r(?x, D) = \mathsf{pos}$. Now, if $\delta \in (\exists p.D)_{\mu(?x)=r})^{\mathcal{I}}$, then δ has at least one p-successor that is an instance of $D_{\mu(?x)=r}$. Since $\mathcal{O} \models r \sqsubseteq s$ and by induction hypothesis, this p-successor is an instance of $D_{\mu(?x)=s}$. Hence, $\delta \in ((\exists p.D)_{\mu(?x)=s})^{\mathcal{I}}$. The proof is analogous for $\mathsf{pol}_r(?x, C) = \mathsf{neg}$.

- For $C = \exists ?x.D$ and $\mathsf{pol}_r(?x, C) = \mathsf{pos}$, we also have $\mathsf{pol}_r(?x, D) = \mathsf{pos}$. Note that $?x$ occurs in D since otherwise the case is handled already above. Now, if $\delta \in ((\exists ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$, then δ has at least one r-successor which is an instance of $D_{\mu(?x)=r}$. Since $\mathcal{O} \models r \sqsubseteq s$ and by induction hypothesis, δ has at least one s-successor that is an instance of $D_{\mu(?x)=s}$. Hence, $\delta \in ((\exists ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$.

- For $C = \forall p.D$ and $\mathsf{pol}_r(?x, C) = \mathsf{pos}$, we also have $\mathsf{pol}_r(?x, D) = \mathsf{pos}$. Now, if $\delta \in ((\forall p.D)_{\mu(?x)=r})^{\mathcal{I}}$, then $\delta \in (\forall p.(D)_{\mu(?x)=r})^{\mathcal{I}}$ and each p-successor of δ is an instance of $D_{\mu(?x)=r}$. Since $\mathcal{O} \models r \sqsubseteq s$ and by induction hypothesis, these p-successors are also instances of $D_{\mu(?x)=s}$. Hence, $\delta \in (\forall p.(D_{\mu(?x)=s}))^{\mathcal{I}} = ((\forall p.D)_{\mu(?x)=s})^{\mathcal{I}}$. The proof is analogous for $\mathsf{pol}_r(?x, C) = \mathsf{neg}$.

- For $C = \forall ?x.D$ and $\mathsf{pol}_r(?x, C) = \mathsf{neg}$, we also have $\mathsf{pol}_r(?x, D) = \mathsf{neg}$. Note that $?x$ occurs in D since otherwise the case is handled already above. Now, if $\delta \in ((\forall ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$, we have to show that $\delta \in ((\forall ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$. In contrary to what is to be shown, assume that $\delta \notin ((\forall ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$, i.e., $\delta \in (\exists r.(\neg D)_{\mu(?x)=r})^{\mathcal{I}}$. Hence, there is some $\delta' \in \Delta^{\mathcal{I}}$ such that $\langle \delta, \delta' \rangle \in r^{\mathcal{I}}$ and $\delta' \in ((\neg D)_{\mu(?x)=r})^{\mathcal{I}}$. Since $\mathcal{O} \models r \sqsubseteq s$, δ' is also an s-successor of δ and, by induction hypothesis, we have $\delta' \in ((\neg D)_{\mu(?x)=s})^{\mathcal{I}}$ which is a contradiction.

- For $C = \geqslant n\ p.D$ and $\mathsf{pol}_r(?x, C) = \mathsf{pos}$, if $\delta \in ((\geqslant n\ p.D)_{\mu(?x)=r})^{\mathcal{I}}$, then δ has at least n distinct p-successors that are instances of $D_{\mu(?x)=r}$. Since $\mathcal{O} \models r \sqsubseteq s$ and by induction hypothesis, these p-successors are also instances of $D_{\mu(?x)=s}$. Hence, $\delta \in ((\geqslant n\ p.D)_{\mu(?x)=s})^{\mathcal{I}}$. The proof is analogous for $\mathsf{pol}_r(?x, C) = \mathsf{neg}$

- For $C = \geqslant n\ ?x.D$ and $\mathsf{pol}_r(?x, C) = \mathsf{pos}$, we also have $\mathsf{pol}_r(?x, D) = \mathsf{pos}$. Note that $?x$ occurs in D since otherwise the case is handled already above. Now, if $\delta \in ((\geqslant n\ ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$, then δ has at least n distinct r-successors which are instances of $D_{\mu(?x)=r}$. Since $\mathcal{O} \models r \sqsubseteq s$ and by induction hypothesis, δ has at least n distinct s-successors that are instances of $D_{\mu(?x)=s}$. Hence, $\delta \in ((\geqslant n\ ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$.

- For $C = \leqslant n\ p.D$ and $\mathsf{pol}_r(?x, C) = \mathsf{pos}$, if $\delta \in ((\leqslant n\ p.D)_{\mu(?x)=r})^{\mathcal{I}}$, we have to show that $\delta \in ((\leqslant n\ p.D)_{\mu(?x)=s})^{\mathcal{I}}$. Note that $\mathsf{pol}_r(?x, D) = \mathsf{neg}$. In contrary to what is to be shown, assume that $\delta \in (\neg(\leqslant n\ p.D)_{\mu(?x)=s})^{\mathcal{I}}$, i.e., $\delta \in ((\geqslant n+1\ p.D)_{\mu(?x)=s})^{\mathcal{I}}$. Hence, δ has at least $n+1$ distinct p-successors which are instances of $D_{\mu(?x)=s}$. Since $\mathsf{pol}_r(?x, D) = \mathsf{neg}$ and by induction hypothesis, these $D_{\mu(?x)=s}$ instances are also $D_{\mu(?x)=r}$ instances and $\delta \in (\geqslant n+1\ p.(D)_{\mu(?x)=r})^{\mathcal{I}} = ((\geqslant n+1\ p.D)_{\mu(?x)=r})^{\mathcal{I}}$, which is a contradiction. The proof is analogous for $\mathsf{pol}_r(?x, C) = \mathsf{neg}$.

- For $C = \leqslant n\ ?x.D$ and $\mathsf{pol}_r(?x, C) = \mathsf{neg}$, we have $\mathsf{pol}_r(?x, D) = \mathsf{pos}$. Note that $?x$ occurs in D since otherwise the case is handled already above. If $\delta \in ((\leqslant n\ ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$ we have to show that $\delta \in ((\leqslant n\ ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$. In contrary to what is to be shown, assume that $\delta \in (\neg(\leqslant n\ ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$, i.e., $\delta \in ((\geqslant n+1\ ?x.D)_{\mu(?x)=r})^{\mathcal{I}}$. Hence, δ has at least $n+1$ distinct r-successors which are instances of $D_{\mu(?x)=r}$. Since $\mathcal{O} \models r \sqsubseteq s$, and by induction hypothesis, these r-successors are also s-successors and instances of $D_{\mu(?x)=s}$. Hence, $\delta \in ((\geqslant n+1\ ?x.D)_{\mu(?x)=s})^{\mathcal{I}}$, which is a contradiction.

- For $C = \ (= n\ ?x.D)$ or $C = \ (= n\ r.D)$, the polarity of $?x$ in C is always positive and negative, so $\mathsf{pol}_r(?x, C)$ is undefined and the case cannot occur.
- The cases for $?x$ occurring in the form of an inverse ($?x^-$) are analogous, given that $\mathcal{O} \models r \sqsubseteq s$ iff $\mathcal{O} \models r^- \sqsubseteq s^-$.

□

Chapter 6

Query Answering Algorithm

In this chapter we present the proposed query answering algorithm together with the developed optimizations and prove that it is a terminating, sound and complete procedure.

6.1 Algorithm Description

Algorithm 4 shows an optimized way of evaluating queries using static ordering. In some places in the algorithm we use μ as a partial function for a query q (atom at) for which Conditions 1, 2 and 3 of Definition 5 (Section 2.3) hold. This appears to be so, because of the way that axiom templates are evaluated (they are evaluated one by one in a row); in an iteration of the algorithm some variables from the template under analysis may have already been bound from a previous iteration (from previous templates in the plan). Hence, when we write $\mu(\mathsf{at})$ in the algorihm we denote the result of replacing each variable $v \in \mathrm{dom}(\mu) \cap \mathsf{Var}(\mathsf{at})$ in at with $\mu(v)$.

First, axiom templates are simplified where possible (rewrite, line 1). The method rewrite takes a query q, iterates over all its templates and rewrites each of them according to Definition 17 of Section 5.1. The rewritten templates form a new query q', which has the same set of answers over $\mathcal{O}$ as q.

Next, the method connectedComponents (line 2) partitions the axiom templates into sets of connected components, i.e., within a component the templates share common variables, whereas between components there are no shared variables. Unconnected components unnecessarily increase the amount of intermediate results and, instead, one can simply combine the results for the components in the end (line 32).

For each component, we proceed as described below: we first determine an order (method order in line 5). The method order takes an ontology $\mathcal{O}$ and a query q over $\mathcal{O}$, and iterates $|q|$ times finding, at each iteration i, the next template at_i that, according to Definition 12 (Section 4.3), should be added next to the (partial) execution plan $(at_1, \ldots, at_{i-1})$ using the static cost function defined in Section 4.3. The method returns a complete execution plan for q over $\mathcal{O}$.

For a simple axiom template, which contains so far unbound variables, we call a specialized reasoner method to retrieve entailed results, i.e., mappings for unbound variables (callSpecificReasonerTask in line 10), which has already been described in Section 4.3. For templates with all their variables bound, we check whether the mappings lead to entailed axioms (lines 11 to 14). For all other cases, i.e.,

Algorithm 4 evaluate$(\mathcal{O}, q)$

Input: $\mathcal{O}$: the queried $\mathcal{SROIQ}$ ontology
q: a query over $\mathcal{O}$
Output: a set of solutions for evaluating q over $\mathcal{O}$

1: $q' :=$ rewrite(q)
2: $q^1, \ldots, q^m :=$connectedComponents(q')
3: **for** j=1, . . . , m **do**
4: $S_j := \{\mu_0 \mid \mathrm{dom}(\mu_0) = \emptyset\}$
5: $\mathsf{at}_1, \ldots, \mathsf{at}_n :=$ order$(\mathcal{O}, q^j)$
6: **for** $i = 1, \ldots, n$ **do**
7: $R := \emptyset$
8: **for each** $\mu \in S_j$ **do**
9: **if** isSimple(at_i) **and** Var$(\mathsf{at}_i) \setminus \mathrm{dom}(\mu) \neq \emptyset$ **then**
10: $R := R \cup \{\mu' \cup \mu \mid \mu' \in$ callSpecificReasonerTask$(\mu(\mathsf{at}_i))\}$
11: **else if** Var$(\mathsf{at}_i) \setminus \mathrm{dom}(\mu) = \emptyset$ **then**
12: **if** $\mathcal{O} \models \mu(\mathsf{at}_i)$ **then**
13: $R := R \cup \{\mu\}$
14: **end if**
15: **else**
16: $V_{\mathrm{opt}} := \{?x \mid ?x \notin \mathrm{dom}(\mu),$ Theorem 1 or 2 applies to $?x$ and $\mathsf{at}_i\}$
17: $B :=$ initializeVariableMappings$(\mathcal{O}, \mathsf{at}_i, \mu, V_{\mathrm{opt}})$
18: **while** $B \neq \emptyset$ **do**
19: $\mu' :=$ removeMapping(B)
20: **if** $\mathcal{O} \models \mu'(\mathsf{at}_i)$ **then**
21: $R := R \cup \{\mu'\}$
22: **for each** $?x \in V_{\mathrm{opt}}$ **do**
23: $B := B \cup$ getPossibleMappings$(\mathcal{O}, ?x, \mathsf{at}_i, \mu')$
24: **end for**
25: **end if**
26: **end while**
27: **end if**
28: **end for**
29: $S_j := R$
30: **end for**
31: **end for**
32: $R_{\mathrm{ans}} := \{\mu_1 \cup \ldots \cup \mu_m \mid \mu_j \in S_j, 1 \leq j \leq m\}$
33: **return** R_{ans}

Algorithm 5 initializeVariableMappings($\mathcal{O}, \mathsf{at}, \mu, V_{\text{opt}}$)

Input: $\mathcal{O}$: the queried $\mathcal{SROIQ}$ ontology
at: an axiom template
μ: a partial mapping
V_{opt}: the variables of at to which Theorem 1 or 2 applies
Output: a set of mappings

```
1:  S := {μ}
2:  for each ?x ∈ Var(at) \ dom(μ) do
3:    R := ∅
4:    if ?x ∈ V_C and ?x ∈ V_opt then
5:      for each μ' ∈ S do
6:        if pol_c(?x, at) = pos then
7:          μ'(?x) := ⊤
8:        else
9:          μ'(?x) := ⊥
10:       end if
11:       R := R ∪ {μ'}
12:     end for
13:   else if ?x ∈ V_R and ?x ∈ V_opt then
14:     for each μ' ∈ S do
15:       if pol_r(?x, at) = pos then
16:         μ'(?x) := ⊤_r
17:       else
18:         μ'(?x) := ⊥_r
19:       end if
20:       R := R ∪ {μ'}
21:     end for
22:   else
23:     R := {μ' | μ'(?x) = a, a ∈ N_C^O or a ∈ N_R^O or a ∈ N_I^O and μ'(?y) = μ_1(?y)
                  for μ_1 ∈ S and ?y ∈ dom(μ_1)}
24:   end if
25:   S := R
26: end for
27: return S
```

for complex axiom templates with unbound variables, we check which compatible mappings yield an entailed axiom (lines 15 to 27). In particular, we first initialize a set B of candidate mappings for the unbound variables of the axiom template (line 17, which refers to Algorithm 5). Algorithm 5 initializes the unbound variables of axiom templates on which Theorem 1 or 2 of Section 5.2 applies to $\top$ ($\top_r$) or $\bot$ ($\bot_r$) depending on whether the respective polarity function returns pos or neg (see Definitions 18 and 19 in Section 5.2). For template variables on which the optimization is not applicable, all compatible mappings are returned (line 23 of Algorithm 5). The method removeMapping (line 19) returns a mapping μ' from B deleting this mapping from B. From an implementation point of view, B has been implemented as a FIFO structure and the method removeMapping returns (and removes from B) the first mapping in the structure. We then instantiate the axiom template using μ' and check entailment. In case the entailment holds, we

Algorithm 6 getPossibleMappings($\mathcal{O}, ?x, \mathsf{at}, \mu$)

Input: $\mathcal{O}$: the queried $\mathcal{SROIQ}$ ontology
$?x$: a concept or role variable
at: an axiom template in which $?x$ occurs
μ: a mapping with $?x \in \mathrm{dom}(\mu)$
Output: a set of mappings

1: $S := \emptyset$
2: **if** $?x \in V_C$ **then**
3: **if** $\mathsf{pol}_c(?x, \mathsf{at}) = \mathsf{pos}$ **then**
4: $S := \{\mu' \mid \mu'(?x) = A, A$ is a direct sub-concept of $\mu(?x)$ in $\mathcal{O}$, $\mu'(?y) = \mu(?y)$ for $?y \in \mathrm{dom}(\mu) \setminus \{?x\}\}$
5: **else**
6: $S := \{\mu' \mid \mu'(?x) = A, A$ is a direct super-concept of $\mu(?x)$ in $\mathcal{O}$, $\mu'(?y) = \mu(?y)$ for $?y \in \mathrm{dom}(\mu) \setminus \{?x\}\}$
7: **end if**
8: **else**
9: **if** $\mathsf{pol}_r(?x, \mathsf{at}) = \mathsf{pos}$ **then**
10: $S := \{\mu' \mid \mu'(?x) = r, r$ is a direct sub-role of $\mu(?x)$ in $\mathcal{O}$, $\mu'(?y) = \mu(?y)$ for $?y \in \mathrm{dom}(\mu) \setminus \{?x\}\}$
11: **else**
12: $S := \{\mu' \mid \mu'(?x) = r, r$ is a direct super-role of $\mu(?x)$ in $\mathcal{O}$, $\mu'(?y) = \mu(?y)$ for $?y \in \mathrm{dom}(\mu) \setminus \{?x\}\}$
13: **end if**
14: **end if**
15: **return** S

first extend the set R with the current mapping μ' and we afterwards extend the set B of possible mappings for the variables to which the hierarchy optimization of Section 5.2 is applicable (getPossibleMappings in line 23). For example, if we just checked a mapping μ that maps a concept variable $?x$ to the concept name A and $?x$ only occurs positively in the axiom template, then we add to the set B all mappings that map $?x$ to a direct sub-concept[1] of A (see Algorithm 6, line 4). We then repeat the procedure until B is empty (lines 18 to 26).

In the implementation we use a more involved procedure for the while loop (lines 18 to 26) , i.e., in order to avoid checking entailment of an instantiated axiom template more than once with the same mapping, which can be the case with the concept (role) hierarchy traversal that we perform as we have explained in the example in Section 5.2, we keep track of already processed mappings and check only those that have not been checked in a previous iteration of the while loop. As a further optimization we directly (i.e., without performing a consistency check) add to the set of solution mappings, mappings that are equivalent[2] to mappings that are already

[1] We say that a concept name A is a direct sub-concept of a concept name B w.r.t. $\mathcal{O}$, if $\mathcal{O} \models A \sqsubseteq B$ and there is no other concept name A' such that $\mathcal{O} \models A' \sqsubseteq B$, $\mathcal{O} \models A \sqsubseteq A'$ and $\mathcal{O} \not\models A \equiv A'$. In a similar way we can define the direct super-concept, the direct sub-role and direct super-role.

[2] Let μ and μ' be two mappings for a query q over an ontology $\mathcal{O}$ ($\mathrm{dom}(\mu) = \mathrm{dom}(\mu')$). We say that μ and μ' are *equivalent mappings* if for each $?x \in (\mathrm{dom}(\mu) \cap (V_C \cup V_R))$ it holds $\mu(?x) \equiv \mu'(?x)$ and for each $?y \in (\mathrm{dom}(\mu) \cap V_I)$ it holds $\mu(?y) \approx \mu'(?y)$

Table 6.1: *Queried Ontology*

$$\mathcal{T} = \begin{array}{lll} \{A_4 \sqsubseteq A_1 & A_6 \sqsubseteq A_3 & A_5 \sqsubseteq A_2 \\ A_8 \sqsubseteq A_4 & A_7 \sqsubseteq A_3 & A_{10} \sqsubseteq A_5 \\ A_9 \sqsubseteq A_4 & A_{11} \sqsubseteq A_6 & A_4 \sqsubseteq \geq 2r.A_{11}\} \end{array} \quad \mathcal{A} = \begin{array}{l} \{A_2(a)\ A_2(b) \\ A_1(b)\ A_1(c) \\ (A_1 \sqcup A_3)(a)\} \end{array}$$

in the set of solution mappings, since such mappings are also answers. For ease of presentation, the above described optimizations are not shown in Algorithm 4.

For the dynamic ordering, Algorithm 4 has to be changed as follows: We first compute the number of axiom templates in the currently evaluated connected component q^j; i.e., $n := |q^j|$. We then swap line 5 and line 6, i.e., instead of ordering all axiom templates before the loop that evaluates the axiom templates, we order within the for loop. The function order gets as additional input parameters the set of currently computed solutions S_j and the already evaluated (partial) execution plan $(\mathsf{at}_1, \ldots, \mathsf{at}_{i-1})$ if $i > 1$ else the empty sequence if there is no plan so far; i.e., if $i = 1$. It returns the next cheapest axiom template according to Definition 12 and using the dynamic ordering function defined in Section 4.3. Hence, we have $\mathsf{at}_i := \mathsf{order}(\mathcal{O}, q^j, S_j, (at_1, \ldots at_{i-1}))$ if $i > 1$, else $\mathsf{at}_1 := \mathsf{order}(\mathcal{O}, q^j, S_j, ())$, instead of $\mathsf{at}_1, \ldots, \mathsf{at}_n := \mathsf{order}(\mathcal{O}, q^j)$. We further insert a line after calling order to remove the cheapest axiom template from the current component: $q^j := q^j \setminus \{\mathsf{at}_i\}$. As a result, the next iteration of the for loop (lines 6- 30) will compute the cheapest axiom template amongst the not yet evaluated templates until, in the last iteration, we only have one axiom template left.

Note that the presented algorithm is a general algorithm in which any reasoner that implements the callSpecificReasonerTask method can be used. If the used reasoner additionally provides statistics for query ordering that are more accurate w.r.t. the way that it implements the reasoning tasks then a good ordering can be obtained. Note also that the reduction in the number of consistency checks for complex axiom templates, when the hierarchy optimization of Section 5.2 is used, is not taken into account by the cost functions in the ordering procedure, since it is difficult to estimate this reduction beforehand; i.e., before query evaluation takes place.

We next show a run of Algorithm 4 for the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ shown in Table 6.1, where A_i for $1 \leq i \leq 11$ are concept names, r a role name and a,b,c individual names. In the beginning, before a system accepts queries, a check is usually performed to determine whether the queried ontology is consistent. The ontology $\mathcal{O}$ is consistent and let us assume that the sets $K[A_2] = \{a, b\}$, $P[A_2] = \emptyset$, $K[A_1] = \{b, c\}$, $P[A_1] = \{a\}$ have been constructed during this check (the known and possible sets for all other concepts and roles are empty). Moreover, we assume that $\mathcal{O}$ has been classified; in Figure 6.1 we show the concept hierarchy.

Let $q = \{A_1(?x), A_2(?x), ?y \sqsubseteq A_4, ?y \sqsubseteq \exists r.?z\}$ be a query. The method rewrite does not have any effect on q and hence $q' = q$. The query q' is split to two connected components by method connectedComponents, i.e., $q^1 = \{A_1(?x), A_2(?x)\}$ and $q^2 = \{?y \sqsubseteq A_4, ?y \sqsubseteq \exists r.?z\}$. For q^1 the method order returns the execution plan $(A_2(?x), A_1(?x))$, since $C_E > C_L$ and

$$\mathsf{Ec}^s_{A_2(?x)} + \mathsf{Rs}^s_{A_2(?x)} = 2 \cdot d \cdot C_L + 2 < (2 \cdot d \cdot C_L + d \cdot C_E) + (2 + P_{IS}) = \mathsf{Ec}^s_{A_1(?x)} + \mathsf{Rs}^s_{A_1(?x)}$$

where C_L, C_E are the look-up and entailment costs, P_{IS} is the possible instance success, $\mathsf{Ec}^s_{\mathsf{at}}$ is the cost of the reasoning task required to evaluate at and $\mathsf{Rs}^s_{\mathsf{at}}$ is the result output size that the evaluation of at incurs, using the static cost function

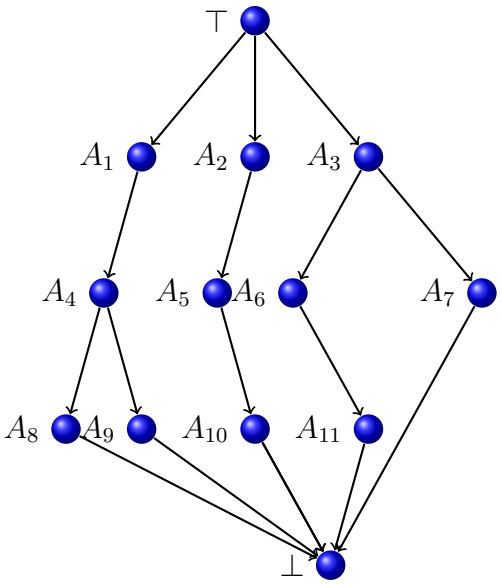

Figure 6.1: *Concept hierarchy example*

defined in Section 4.3 and d is the depth of the concept hierarchy. In the first iteration of the for loop (lines 6-30), since, according to Definition 8 (Section 4.1), $A_2(?x)$ is a simple template, $S_1 = \{\mu_0\}$, $R = \emptyset$ and $\mathsf{Var}(A_2(?x)) \setminus \mathrm{dom}(\mu_0) = \{?x\}$, the method callSpecificReasonerTask returns a set R, which contains mappings of $?x$ to the known instances a, b of A_2 (doing look-ups in the cached sets of known and possible instances of A_2), i.e., $R = \{\{?x \mapsto a\}, \{?x \mapsto b\}\}$. From line 29, $S_1 = R$. In the second iteration of the for loop, $R = \emptyset$ and the template $A_1(?x)$ is under consideration. The mapping $\mu = \{?x \mapsto a\}$ in S_1 is first considered for the template $A_1(?x)$. Since $A(?x)$ is a simple template, $\mathsf{Var}(A_1(?x)) \setminus \mathrm{dom}(\mu) = \emptyset$ and $\mathcal{O} \not\models A_1(a)$ (a consistency check is performed for determining whether the possible instance a of A_1 is a real instance of A_1), $R = \emptyset$. Afterwards, the mapping $\mu = \{?x \mapsto b\}$ in S_1 is considered for the simple template $A_1(?x)$ and since $\mathsf{Var}(A_1(?x)) \setminus \mathrm{dom}(\mu) = \emptyset$ and $\mathcal{O} \models A_1(b)$ (this is determined by a cache look-up), $R = \{?x \mapsto b\}$ and $S_1 = R = \{?x \mapsto b\}$. In the next iteration of the for loop (lines 3-31) the component q^2 is under consideration. The method order returns the execution plan $(?y \sqsubseteq A_4, ?y \sqsubseteq \exists r.?z)$, since

$$\mathsf{Ec}^s_{?y \sqsubseteq A_4} + \mathsf{Rs}^s_{?y \sqsubseteq A_4} = 13 \cdot C_L + 4 < 13 \cdot 13 \cdot C_E + 13 \cdot 13 = \mathsf{Ec}^s_{?y \sqsubseteq \exists r.?z} + \mathsf{Rs}^s_{?y \sqsubseteq \exists r.?z}$$

In the first iteration of the for loop (lines 6-30), since, according to Definition 8 (Section 4.1), $?y \sqsubseteq A_4$ is a simple template, $S_2 = \{\mu_0\}$, $R = \emptyset$, and $\mathsf{Var}(?y \sqsubseteq A_4) \setminus \mathrm{dom}(\mu_0) = \{?y\}$, the method callSpecificReasonerTask returns a set of mappings R, which contains mappings of $?y$ to sub-concepts of A_4 using look-ups in the cached concept hierarchy. From lines 10 and 29,

$$S_2 = R = \{\{?y \mapsto A_4\}, \{?y \mapsto A_8\}, \{?y \mapsto A_9\}, \{?y \mapsto \bot\}\}.$$

In the second iteration of the for loop (lines 6-30), the template $?y \sqsubseteq \exists.r?z$ is under consideration. In lines 8-28 we iterate over every $\mu \in S_2$. We will show how the algorithm runs for $\mu = \{?y \mapsto A_4\}$. The run of the for loop (lines 8-28) for the other mappings in S_2 is similar. From line 16, $V_{opt} = \{?z\}$ and $B =$

initializeVariableMappings($\mathcal{O}, ?y \sqsubseteq \exists.r?z, \mu, \{?z\}$). It holds $?z \in \mathsf{Var}(?y \sqsubseteq \exists r.?z) \setminus \{?y\}$, $?z \in V_C \cap V_{opt}$ and $\mathsf{pol}_c(?z, ?y \sqsubseteq \exists r.?z) = \mathsf{pos}$. Hence, from lines 11 and 25 of Algorithm 5 $S = R = \{\{?y \mapsto A_4, ?z \mapsto \top\}\}$ and, hence, from Algorithm 4, $B = \{\{?y \mapsto A_4, ?z \mapsto \top\}\}$. From line 19, the only mapping, let us say μ' is removed from B. Since $\mathcal{O} \models A_4 \sqsubseteq \exists r.\top$, $R = \{\{?y \mapsto A_4, ?z \mapsto \top\}\}$ and for $?z \in V_{opt}$, $B =$ getPossibleMappings($\mathcal{O}, ?z, ?y \sqsubseteq \exists.r?z, \mu'$) from line 23. Since $?z \in V_C$ and $\mathsf{pol}_c(?z, ?y \sqsubseteq \exists r.?z) = \mathsf{pos}$,

$$B = \{\{?y \mapsto A_4, ?z \mapsto A_1\}, \{?y \mapsto A_4, ?z \mapsto A_2\}, \{?y \mapsto A_4, ?z \mapsto A_3\}\}$$

In the next iteration of the while loop in lines 18-26, the first mapping $\mu' = \{?y \mapsto A_4, ?z \mapsto A_1\}$ is considered and removed from B for which holds $\mathcal{O} \not\models A_4 \sqsubseteq \exists r.A_1$. Afterwards, we choose the next mapping in B, i.e., $\mu' = \{?y \mapsto A_4, ?z \mapsto A_2\}$ for which again holds $\mathcal{O} \not\models A_4 \sqsubseteq \exists r.A_2$. In the end, we choose the mapping $\mu' = \{?y \mapsto A_4, ?z \mapsto A_3\}$. Since $\mathcal{O} \models A_4 \sqsubseteq \exists r.A_3$, $R = \{\{?y \mapsto A_4, ?z \mapsto \top\}, \{?y \mapsto A_4, ?z \mapsto A_3\}\}$ and $B = \{\{?y \mapsto A_4, ?z \mapsto A_6\}, \{?y \mapsto A_4, ?z \mapsto A_7\}\}$. In the next iteration of the while loop we choose $\mu' = \{?y \mapsto A_4, ?z \mapsto A_6\}$. Since $\mathcal{O} \models A_4 \sqsubseteq \exists r.A_6$, the sets R (in line 21 of Algorithm 4) and B (from Algorithm 6) are updated to

$$\begin{aligned} R =& \{\{?y \mapsto A_4, ?z \mapsto \top\}, \{?y \mapsto A_4, ?z \mapsto A_3\}, \{?y \mapsto A_4, ?z \mapsto A_6\}\} \text{ and} \\ B =& \{\{?y \mapsto A_4, ?z \mapsto A_7\}, \{?y \mapsto A_4, ?z \mapsto A_{11}\}\}. \end{aligned}$$

In the same way, the mapping $\mu' = \{?y \mapsto A_4, ?z \mapsto A_7\}$ does not lead to an entailed by $\mathcal{O}$ axiom and, hence, is not added to the set R. The mapping $\mu' = \{?y \mapsto A_4, ?z \mapsto A_{11}\}$ leads to an entailed by $\mathcal{O}$ axiom and, hence, is added to the set R, i.e.,

$$R = \{\{?y \mapsto A_4, ?z \mapsto \top\}, \{?y \mapsto A_4, ?z \mapsto A_3\}, \{?y \mapsto A_4, ?z \mapsto A_6\}, \{?y \mapsto A_4, ?z \mapsto A_{11}\}\}$$

Finally, the mapping $\{?y \mapsto A_4, ?z \mapsto \bot\}$ does not lead to an entailed by $\mathcal{O}$ axiom and, hence, is not added to the set R. In a similar way the else statement of lines 15-27 is run for each of the other mappings in S_2. In fact, the mappings tested for the other two mappings in S_2, i.e., $\{?y \mapsto A_8\}$ and $\{?y \mapsto A_9\}$ are exactly the same with those described above for the mapping $\{?y \mapsto A_4\}$, if we map the variable $?y$ to A_8 or A_9 instead of A_4. Similarly, the mapping $\{?y \mapsto \bot\}$ leads to entailed axioms for every concept used as a mapping for $?z$. Hence, the set S_2 in line 29 contains the mappings:

$$\begin{aligned} S_2 = \{&\{?y \mapsto A_4, ?z \mapsto \top\}, \{?y \mapsto A_4, ?z \mapsto A_3\}, \{?y \mapsto A_4, ?z \mapsto A_6\}, \{?y \mapsto A_4, ?z \mapsto A_{11}\} \\ &\{?y \mapsto A_8, ?z \mapsto \top\}, \{?y \mapsto A_8, ?z \mapsto A_3\}, \{?y \mapsto A_8, ?z \mapsto A_6\}, \{?y \mapsto A_8, ?z \mapsto A_{11}\} \\ &\{?y \mapsto A_9, ?z \mapsto \top\}, \{?y \mapsto A_9, ?z \mapsto A_3\}, \{?y \mapsto A_9, ?z \mapsto A_6\}, \{?y \mapsto A_9, ?z \mapsto A_{11}\} \\ &\{?y \mapsto \bot, ?z \mapsto \top\}, \{?y \mapsto \bot, ?z \mapsto A_1\}, \{?y \mapsto \bot, ?z \mapsto A_2\}, \{?y \mapsto \bot, ?z \mapsto A_3\} \\ &\{?y \mapsto \bot, ?z \mapsto A_4\}, \{?y \mapsto \bot, ?z \mapsto A_5\}, \{?y \mapsto \bot, ?z \mapsto A_6\}, \{?y \mapsto \bot, ?z \mapsto A_7\} \\ &\{?y \mapsto \bot, ?z \mapsto A_8\}, \{?y \mapsto \bot, ?z \mapsto A_9\}, \{?y \mapsto \bot, ?z \mapsto A_{10}\}, \{?y \mapsto \bot, ?z \mapsto A_{11}\} \\ &\{?y \mapsto \bot, ?z \mapsto \bot\}\} \end{aligned}$$

The output R_{ans} of Algorithm 4 is the cartesian product of the mappings in S_1 and S_2 as shown in line 32.

6.2 Termination, Soundness and Completeness

In this section we prove that the Algorithm 4 terminates and is sound and complete as long as the used reasoning algorithm is sound and complete.

First we prove that Algorithm evaluate terminates.

Theorem 3. *(**Termination**) Algorithm* evaluate *terminates.*

Proof. The query q has a finite number of axiom templates. The methods rewrite, connectedComponents, order are all terminating procedures. The method rewrite iterates over the axiom templates of q and rewrites each to a set of equivalent axiom templates according to Definition 17. The method connectedComponents splits the rewritten query to a finite number, m, of connected components while the method order orders the templates as explained in Section 4.3. The method callSpecificReasonerTask is assumed to be a terminating procedure and returns a finite number of mappings. The method initializeVariableMappings returns a finite number of mappings that are assigned to the set B in line 17. In this method variables of the atom at under analysis that are not in the domain of the current mapping μ are initialized, i.e., unbound variables of at (i.e., variables of $\mu(\mathsf{at})$) to which Theorem 1 or 2 (Section 5.2) is applied, are mapped either to $\top$, $\bot$, $\top_r$, or $\bot_r$ and concept, role and individual variables to which Theorem 1 or 2 does not apply are mapped to corresponding values from the finite set of concept, role and individual names that appear in $\mathcal{O}$. At each iteration of the while loop in lines 18-26, a mapping $\mu \in B$ is examined and removed from B and the set B is (possibly) extended by the set of mappings that the method getPossibleMappings returns. The method getPossibleMappings creates new mappings by changing values of concept or role variables in μ to direct sub-concepts, super-concepts, sub-roles or super-roles while traversing, at each iteration, the concept and role hierarchies top down or bottom up (depending on the polarity of variables). Since the concept and role hierarchies are finite, the set B will contain a finite number of mappings at each iteration and hence the while loop will be repeated a finite number of times. □

In the following, we prove that algorithm evaluate is sound and complete. Let R_{ans} be the set of solution mappings that algorithm evaluate returns. By sound we mean that $R_{ans} \subseteq \mathsf{ans}(\mathcal{O}, q)$. By complete we mean that $\mathsf{ans}(\mathcal{O}, q) \subseteq R_{ans}$.

Theorem 4. *(**Soundness**) Let q be a query, $\mathcal{O}$ an ontology and R_{ans} the output of Algorithm 4. It holds that if $\mu \in R_{ans}$ then $\mu \in \mathsf{ans}(\mathcal{O}, q)$.*

Proof. The method rewrite (see Definition 17) does not affect the answers to a query q over $\mathcal{O}$, i.e., $\mathsf{ans}(\mathcal{O}, q) = \mathsf{ans}(\mathcal{O}, \mathsf{rewrite}(q))$. The method connectedComponents does not affect the answers of q; it just splits the query into several components (line 2) that are evaluated separately (lines 3-31) and the cartesian product of the answers is then taken (line 32) [30], i.e.,

$$\mathsf{ans}(\mathcal{O}, q) = \bigcup_{j=1}^{m} \mathsf{ans}(\mathcal{O}, q_j).$$

Hence, it is enough to show that the set of mappings S_j (line 29) that is the output of lines 4-30 is indeed a solution set for $q = q_j$ over $\mathcal{O}$.

The method order (line 5) does not change the query in any way; it just orders the axiom templates as explained in Section 4.3, i.e., $\mathsf{ans}(\mathcal{O}, q) = \mathsf{ans}(\mathcal{O}, \mathsf{order}(\mathcal{O}, q))$. Hence, we afterwards show that the algorithm in lines 6-30 is sound for $q = q_j$ over $\mathcal{O}$. In the following, with R_i we denote the value of R after the i^{th} iteration of the for loop in lines 6-30 and R_n is the set of solution mappings that the for loop of lines 6-30 of Algorithm 4 gives. We prove that for any mapping μ such that $\mu \in R_n$, it holds $\mu \in \mathsf{ans}(\mathcal{O}, q)$, i.e., $\mu \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1 \dots \mathsf{at}_n\})$ (for every template $\mathsf{at} \in q$, $\mathcal{O} \models \mu(\mathsf{at})$ assuming that the used reasoner is sound). We prove this by induction on the number of iterations of the for loop in lines 6-30 .

Base case: For at_1 in the beginning of the for loop (lines 6-30), $S_j = \{\mu_0\}, R_0 = \emptyset$. We show that for any mapping μ' if $\mu' \in R_1$ then $\mu' \in \mathsf{ans}(\mathcal{O}, \mathsf{at}_1)$. There are three cases for at_1:

1. The atom at_1 is simple and it has free variables (variables not mapped by $\mu_0 \in R_0$). Then the algorithm calls an appropriate task of a sound (and complete) reasoner to retrieve the solution mappings S for the variables of $\mu_0(\mathsf{at}_1)$, i.e., the variables of at_1, and, hence, $R_1 = S$ and $S \in \mathsf{ans}(\mathcal{O}, \mathsf{at}_1)$ (due to reasoner soundness).

2. The atom at_1 does not contain free variables (variables not mapped by μ_0), which, in this case, means that at_1 does not contain variables, (i.e., it is a Boolean atom). In this case, $\mu' = \mu_0$. Since a sound reasoner is used to decide whether $\mathcal{O} \models \mu'(\mathsf{at}_1)$, $\mu' \in R_1$ only if $\mu' \in \mathsf{ans}(\mathcal{O}, \mathsf{at}_1)$.

3. The atom at_1 is a complex atom which contains free variables (variables not mapped by μ_0). Since $\mu' \in R_1$ it was added by line 21 of the algorithm. Since we again assume that a sound reasoner is used to decide whether $\mathcal{O} \models \mu'(\mathsf{at}_1)$, this means $\mu' \in \mathsf{ans}(\mathcal{O}, \mathsf{at}_1)$.

Inductive step: Let R_k be the output of the for loop at lines 6-30 at iteration k with $1 \leq k < n$. Let us assume that μ is a mapping such that $\mu \in R_k$ and $\mu \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \dots, \mathsf{at}_k\})$ (IH). We will show that if $\mu' \in R_{k+1}$ then $\mu' \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \dots, \mathsf{at}_{k+1}\})$. Since $\mu' \in R_{k+1}$, this means that a mapping $\mu \in R_k$ caused its production through lines 8-28 of Algorithm 4 while processing the atom at_{k+1} of q. There are three cases:

1. The atom at_{k+1} is simple and it has free variables (variables not mapped by $\mu \in R_k$). Then the algorithm calls an appropriate task of a sound (and complete) reasoner to retrieve the solution mappings S for the variables of $\mu(\mathsf{at}_{k+1})$. Since $\mu' \in R_{k+1}$, this means that there is a mapping $\mu'' \in S$ such that $\mu' = \mu'' \cup \mu$ from the way R is constructed in line 10. Since $\mu'' \in S$ this means $\mathcal{O} \models \mu''(\mu(\mathsf{at}_{k+1}))$ (μ'' maps variables of at_{k+1} that were not mapped in μ, i.e., the domains of μ and μ'' are disjoint) and, since $\mu' = \mu'' \cup \mu$, it holds $\mathcal{O} \models \mu'(\mathsf{at}_{k+1})$. From induction hypothesis $\mu \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \dots \mathsf{at}_k\})$, and since the domains of μ and μ'' are disjoint and $\mu' = \mu \cup \mu''$, $\mu' \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \dots, \mathsf{at}_{k+1}\})$.

2. The atom at_{k+1} does not contain free variables (variables not mapped by $\mu \in R_k$). Since $\mu' \in R_{k+1}$ this means that $\mathcal{O} \models \mu'(\mathsf{at}_{k+1})$ and $\mu' = \mu$ holds from lines 11-14 of the algorithm. From induction hypothesis $\mu \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \dots, \mathsf{at}_k\})$ and since $\mu' = \mu$ and a sound reasoner is used to decide whether $\mathcal{O} \models \mu'(at_{k+1})$, it holds $\mu' \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \dots, \mathsf{at}_{k+1}\})$.

3. The atom at_{k+1} is a complex atom which contains free variables (variables not mapped by $\mu \in R_k$). Since $\mu' \in R_{k+1}$ it was added by line 21 of the algorithm whose precondition is $\mathcal{O} \models \mu'(\mathsf{at}_{k+1})$. The mapping μ' is a mapping in the set B and, according to lines 15-27, has been produced either i) directly by the call of the method $\mathsf{initializeVariableMappings}(\mathcal{O}, \mathsf{at}_{k+1}, \mu, V_{opt})$ with $\mu \in R_k$ (line 17) or ii) indirectly by a sequence of calls of the method $\mathsf{getPossibleMappings}$ which was initially called for a mapping μ_1 in the set produced by $\mathsf{initializeVariableMappings}(\mathcal{O}, \mathsf{at}_{k+1}, \mu, V_{opt})$, where $\mu \in R_k$ (lines 18-26). Hence, we need to first prove the following Lemma, in which with B_r we denote the set B in the end of the r$^{\text{th}}$ iteration of the while loop.

Lemma 3. *Let n be $|q|$, $\mu \in R_k$ with $1 \leq k < n$, at_{k+1} the atom under analysis at the $(k+1)^{ith}$ iteration of the* for *loop (lines 6-30), l the number of runs of the* while *loop in lines 18-26, when the mapping μ is chosen (line 8) and $\mu' \in B_r$ with $0 \leq r < l$. For μ and μ' it holds $\mu'_{|dom(\mu)} = \mu$.*

Proof. We prove the above claim, by induction on the number of runs of the while loop in lines 18-26 for $\mu \in R_k$.

Base case: For B_0, it holds $B_0 = \mathsf{initializeVariableMappings}(\mathcal{O}, \mathsf{at}_{k+1}, \mu, V_{opt})$, where V_{opt} are the variables of $\mu(\mathsf{at}_{k+1})$ to which Theorem 1 or 2 of Section 5.2 applies. The method $\mathsf{initializeVariableMappings}$ iterates over the variables of $\mu(\mathsf{at}_{k+1})$ and creates mappings for them. In lines 4-21 of Algorithm 5 mappings for the unbound concept (role) variables of $\mu(\mathsf{at}_{k+1})$ to which Theorem 1 or Theorem 2 applies are created and in line 23 mappings for the concept, role or individual variables of $\mu(\mathsf{at}_{k+1})$ to which Theorem 1 and 2 does not apply are created. In other words, for every mapping $\mu_1 \in B_0$, $\mu_1 = \mu \cup \mu_2$ where $\mu \in R_k$ is the mapping with which $\mathsf{initializeVariableMappings}$ is called and $\mathrm{dom}(\mu_2) = \mathsf{Var}(\mathsf{at}_{k+1}) \setminus \mathrm{dom}(\mu)$, which means that the domains of μ and μ_2 are disjoint. Hence, it holds $\mu_{1|\mathrm{dom}(\mu)} = \mu$.

Inductive step: Let us assume that $\mu \in R_k$ with $1 \leq k < n$ and $\mu'_{|\mathrm{dom}(\mu)} = \mu$ holds for every mapping $\mu' \in B_r$ with $0 \leq r < l$ (IH). We will show that $\mu''_{|\mathrm{dom}(\mu)} = \mu$ holds for every mapping $\mu'' \in B_{r+1}$. There are two cases for a mapping $\mu'' \in B_{r+1}$: i) $\mu'' \in B_r$ and it was not chosen (and removed) by method $\mathsf{removeMapping}$ at the r^{th} iteration for which case $\mu''_{|\mathrm{dom}(\mu)} = \mu$ from IH and ii) μ'' was created and added to B_{r+1} in line 23 by method $\mathsf{getPossibleMappings}$. The method $\mathsf{getPossibleMappings}$ takes a mapping $\mu' \in B_r$ and a concept (role) variable $?x \in V_{opt}$ to which Theorem 1 or Theorem 2 applies and which was not bound by μ (was not in the domain of μ) and it maps $?x$ to a different value in Algorithm 6, i.e., a direct sub-concept (line 4) or super-concept (line 6) (direct sub-role (line 10) or super-role (line 12)) of the current concept (role) name producing the set B_{r+1}. This means that the mappings for the variables in $\mathrm{dom}(\mu)$ do not change in any way in B_{r+1} and, hence, for every $\mu'' \in B_{r+1}$ it holds $\mu''_{|\mathrm{dom}(\mu)} = \mu$. □

Since $\mathcal{O} \models \mu'(\mathsf{at}_{k+1})$ and the reasoner used to decide entailment is sound and by Lemma 3, $\mu'_{|\mathrm{dom}(\mu)=\mu}$ we get by IH that $\mu' \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \ldots \mathsf{at}_{k+1}\})$

Note that the optimization according to which mappings that are equivalent to already checked solution mappings are directly added to the answer set does not affect soundness; this follows from the following, easy to prove, lemma.

Lemma 4. *Let $\mathcal{O}$ be an ontology, q a query over $\mathcal{O}$, μ, μ' two equivalent mappings for q over $\mathcal{O}$ and $\mu' \in \mathsf{ans}(\mathcal{O}, q)$. Then $\mu \in \mathsf{ans}(\mathcal{O}, q)$.*

□

Theorem 5. *(**Completeness**) Let q be a query, $\mathcal{O}$ an ontology and R_{ans} the output of Algorithm 4. It holds that if $\mu \in \mathsf{ans}(\mathcal{O}, q)$ then $\mu \in R_{ans}$.*

Proof. The method rewrite (see Definition 17) does not affect the answers to a query q over $\mathcal{O}$, i.e., $\mathsf{ans}(\mathcal{O}, q) = \mathsf{ans}(\mathcal{O}, \mathsf{rewrite}(q))$. The method connectedComponents does not affect the answers of q; it just splits the query into several components (line 2) that are evaluated separately (lines 3-31) and we then take the cartesian product of the answers (line 32). Hence, it is enough to show that the set of solution mappings $\mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \ldots, \mathsf{at}_n\})$ is a subset of R_n, i.e., the output of the loop in lines 4-30 at iteration n.

The method order (line 5) does not change the query in any way; it just orders the axiom templates as explained in Section 4.3, i.e., $\mathsf{ans}(\mathcal{O}, q) = \mathsf{ans}(\mathcal{O}, \mathsf{order}(q))$. Hence, we afterwards show that the algorithm in lines 6-30 is complete for $q = q_j$ over $\mathcal{O}$. In the following, with R_i $(0 \leq i \leq n)$ we denote the value of R after the i^{th} iteration of the for loop in lines 6-30. Let R_n be the set of solution mappings that the for loop of lines 6-30 of Algorithm 4 gives. We prove that for any mapping μ such that $\mu \in \mathsf{ans}(\mathcal{O}, q)$, i.e., $\mu \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1 \ldots \mathsf{at}_n\})$ (for every template $\mathsf{at} \in q$, $\mathcal{O} \models \mu(\mathsf{at})$ assuming that the reasoner used to decide entailment is complete) it holds $\mu \in R_n$. We prove this by induction on the number of iterations of the for loop in lines 6-30.

Base case: For at_1 in the beginning of the for loop (lines 6-30), $S_j = \{\mu_0\}, R_0 = \emptyset$. We show that if $\mu' \in \mathsf{ans}(\mathcal{O}, \mathsf{at}_1)$ then $\mu' \in R_1$. There are three cases for at_1.

1. The atom at_1 is simple and it has free variables (variables not mapped by $\mu_0 \in R_0$). Then the algorithm calls an appropriate task of a (sound and) complete reasoner to retrieve the solution mappings S for the variables of $\mu_0(\mathsf{at}_1)$ and from line 10 it holds $R_1 = S$ and, hence, for every mapping $\mu' \in \mathsf{ans}(\mathcal{O}, \mathsf{at}_1)$ it holds $\mu' \in R_1$ due to reasoner completeness.

2. The atom at_1 does not contain free variables (variables not mapped by μ_0), which in this case means that the atom at_1 does not contain variables, (i.e., it is a Boolean atom). In this case, $\mu' = \mu_0$ and, since $\mu' \in \mathsf{ans}(\mathcal{O}, \mathsf{at}_1)$ and since the reasoner used to decide whether $\mathcal{O} \models \mu'(\mathsf{at}_1)$ is complete this means that $\mathcal{O} \models \mu'(\mathsf{at}_1)$ holds, which, by line 13, implies that $\mu' \in R_1$.

3. The atom at_1 is a complex atom which contains free variables (variables not mapped by μ_0). Note that the variables of $\mu_0(\mathsf{at}_1)$ are mapped to values from $\mathcal{O}$, i.e., concept variables are mapped to concept names from $N_C^{\mathcal{O}}$, role variables are mapped to role names from $N_R^{\mathcal{O}}$ and individual variables are mapped to individual names from $N_I^{\mathcal{O}}$. We prove the claim by assuming that it does not hold and by deriving a contradiction. So, let us assume that $\mu' \in \mathsf{ans}(\mathcal{O}, \mathsf{at}_1)$, which means that $\mathcal{O} \models \mu'(\mathsf{at}_1)$ holds (since the reasoner used to decide entailment is complete), but $\mu' \notin R_1$. The method initializeVariableMappings is

called with $\mu_0 \in R_0$, at_1 and V_{opt}, i.e., the variables of $\mu_0(\mathsf{at}_1)$ to which Theorem 1 or Theorem 2 of Section 5.2 applies, iterates and initializes all variables of $\mu_0(\mathsf{at}_1)$. For each such variable $?x$ there are three cases:

- $?x \in V_{opt}$ and $?x \in V_C$ (i.e., $?x$ is an optimization concept variable), in which case variable $?x$ is mapped to $\top$ or $\bot$ depending on whether the polarity of variable $?x$ in at_1 is positive or negative according to Definition 18 (Section 5.2).
- $?x \in V_{opt}$ and $?x \in V_R$ ($?x$ is an optimization role variable), in which case variable $?x$ is mapped to $\top_r$ or $\bot_r$ depending on whether the polarity of variable $?x$ in at_1 is positive or negative according to Definition 19 (Section 5.2).
- $?x \notin V_{opt}$ ($?x$ is not an optimization variable), in which case several mappings are created and added to S. There are three cases:
  - $?x$ is a concept variable in which case mappings are added to S, each of which maps $?x$ to a concept name in $N_C^{\mathcal{O}}$.
  - $?x$ is a role variable in which case mappings are added to S, each of which maps $?x$ to a role name in $N_R^{\mathcal{O}}$.
  - $?x$ is an individual variable in which case mappings are added to S, each of which maps $?x$ to an individual name in $N_I^{\mathcal{O}}$.

Hence, for variables in $\mu_0(\mathsf{at}_1)$ that are not in V_{opt}, all compatible mappings are added to S and returned by method $\mathsf{initializeVariableMappings}(\mathcal{O}, \mathsf{at}_1, \mu_0, V_{opt})$. Variables in V_{opt} are initialized to $\top$, $\bot$, $\top_r$, or $\bot_r$ (i.e., not all compatible mappings are added to S in this case in $\mathsf{initializeVariableMappings}$). Let l be the number of runs of the while loop in lines 18-26 when the mapping $\mu_0 \in S_j$ is chosen (line 8). In each run r of the while loop, with $0 \leq r < l$, one mapping, let us say $\mu'' \in B_r$, is examined and either $\mathcal{O} \models \mu''(\mathsf{at}_1)$ or $\mathcal{O} \not\models \mu''(\mathsf{at}_1)$ (note that this check is performed by a complete reasoner). From Lemma 3, it holds $\mu''_{|\mathrm{dom}(\mu_0)} = \mu_0$.

- if $\mathcal{O} \models \mu''(\mathsf{at}_1)$ then $\mu'' \in R_1$ (line 21 of Algorithm 4) and $B_{r+1} = (B_r \setminus \{\mu''\}) \cup Q$, where $Q = \bigcup_{?x \in V_{opt}} \mathsf{getPossibleMappings}(\mathcal{O}, ?x, \mathsf{at}_1, \mu'')$, i.e., for each $?x \in V_{opt}$ there are four cases:
  - if $?x \in V_C$ and the polarity of $?x$ in at_1 is positive (negative) then, for every concept name A such that $A \in N_C^{\mathcal{O}}$ and A is a direct sub-concept (super-concept) of $\mu''(?x)$, we add the mapping μ_1 to B_{r+1}, where $\mu_1(?y) = \mu''(?y)$ for $?y \in \mathrm{dom}(\mu'') \setminus \{?x\}$ and $\mu_1(?x) = A$.
  - if $?x \in V_R$ and the polarity of $?x$ in at_1 is positive (negative) then, for every role name s such that $s \in N_R^{\mathcal{O}}$ and s is a direct sub-role (super-role) of $\mu''(?x)$, we add the mapping μ_1 to B_{r+1}, where $\mu_1(?y) = \mu''(?y)$ for $?y \in \mathrm{dom}(\mu'') \setminus \{?x\}$ and $\mu_1(?x) = s$.
- if $\mathcal{O} \not\models \mu''(\mathsf{at}_1)$ then $\mu'' \notin R_1$ and $B_{r+1} = B_r \setminus \{\mu''\}$. For each $?x \in V_{opt}$ there are four cases:
  - $?x \in V_C$ and the polarity of $?x$ in at_1 is positive (negative). According to Theorem 1 Condition 1 (Condition 2) of Section 5.2, for

each concept name A such that $A \sqsubseteq \mu''(?x)$ ($\mu''(?x) \sqsubseteq A$) it holds $\mathcal{O} \not\models \mu_1(\mathsf{at}_1)$, where $\mu_1(?y) = \mu''(?y)$ for $?y \in \mathrm{dom}(\mu'') \setminus \{?x\}$ and $\mu_1(?x) = A$ and, hence, mappings to sub-concepts (super-concepts) of $\mu''(?x)$ are pruned.

– $?x \in V_R$ and the polarity of $?x$ in at_1 is positive (negative). According to Theorem 2 Condition 1 (Condition 2) of Section 5.2, for each role name s such that $s \sqsubseteq \mu''(?x)$ ($\mu''(?x) \sqsubseteq s$) it holds $\mathcal{O} \not\models \mu_1(\mathsf{at}_1)$, where $\mu_1(?y) = \mu''(?y)$ for $?y \in \mathrm{dom}(\mu'') \setminus \{?x\}$ and $\mu_1(?x) = s$ and, hence, mappings to sub-roles (super-roles) of $\mu''(?x)$ are pruned.

Hence, for mappings μ'' which lead to $\mathcal{O} \models \mu''(\mathsf{at}_1)$, all mappings of variables in V_{opt} to (direct) sub-concepts or sub-roles (for variables with positive polarity in at_1) or to (direct) super-concepts or super-roles (for variables with negative polarity in at_1) are added to B_{r+1}, else mappings of variables in V_{opt} to sub-concepts or sub-roles (for variables with positive polarity in at_1) or to super-concepts or super-roles (for variables with negative polarity in at_1) are pruned. From the above it follows that, if $\mu' \notin R_1$, then μ' was pruned from the set of (possible) solution mappings either because μ' was tested directly in line 20 of Algorithm 4 by a complete reasoner and $\mathcal{O} \not\models \mu'(\mathsf{at}_1)$, which contradicts our hypothesis, or, because $\mathcal{O} \not\models \mu_2(\mathsf{at}_1)$ and it holds $\mu'(?x) \sqsubseteq \mu_2(?x)$ ($\mu_2(?x) \sqsubseteq \mu'(?x)$) for each variable $?x \in V_{opt}$ with positive (negative) polarity and $\mu'(?y) = \mu_2(?y)$ for every other variable $?y$, from which according to Theorems 1 and 2 (Section 5.2) follows that $\mathcal{O} \not\models \mu'(\mathsf{at}_1)$, which also contradicts our hypothesis.

Inductive step: Let R_k be the output of the for loop in lines 6-28 at iteration k with $1 \leq k < n$. Let us assume that for every μ such that $\mu \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \ldots, \mathsf{at}_k\})$ it holds $\mu \in R_k$ (IH). We show that if $\mu' \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \ldots, \mathsf{at}_{k+1}\})$ then $\mu' \in R_{k+1}$ assuming that a complete reasoner is used for deciding entailment. Since $\mu' \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \ldots \mathsf{at}_{k+1}\})$, this means that $\mu'_{|\mathsf{Var}(\{\mathsf{at}_1, \ldots \mathsf{at}_k\})} \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \ldots, \mathsf{at}_k\})$ or $\mu \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \ldots, \mathsf{at}_k\})$ for $\mu = \mu'_{|\mathsf{Var}(\{\mathsf{at}_1, \ldots \mathsf{at}_k\})}$. From IH we have $\mu \in R_k$. We prove that $\mu' \in R_{k+1}$. There are three cases for at_{k+1}:

1. The atom at_{k+1} is simple and it has free variables (variables not mapped by $\mu \in R_k$). Since the used reasoner is (sound and) complete, the method callSpecificReasonerTask returns all solution mappings S for the atom $\mu(\mathsf{at}_{k+1})$. Hence, it is the case that for every μ'' with $\mu'' \in \mathsf{ans}(\mathcal{O}, \mu(\mathsf{at}_{k+1}))$, $\mu'' \in S$, which means that $\mu' \in R_{k+1}$ with $\mu' = \mu \cup \mu''$ (line 10 of Algorithm 4), where μ, μ'' have disjoint domains.

2. The atom at_{k+1} does not contain free variables (variables not mapped by $\mu \in R_k$). Hence $\mu' = \mu$ and, since $\mu' \in \mathsf{ans}(\mathcal{O}, \mathsf{at}_1, \ldots, \mathsf{at}_{k+1})$, which by reasoner completeness, means that $\mathcal{O} \models \mu'(\mathsf{at}_{k+1})$ holds, this means that $\mu' \in R_{k+1}$ (line 13 of Algorithm 4).

3. The atom at_{k+1} is a complex atom which contains free variables (variables not mapped by $\mu \in R_k$). Let us assume that the statement to be proved does not hold, i.e., $\mu' \in \mathsf{ans}(\mathcal{O}, \{\mathsf{at}_1, \ldots \mathsf{at}_{k+1}\})$ but $\mu' \notin R_{k+1}$. The proof is exactly the same as the proof for the base case if we substitute $\mu_0 \in R_0$ and at_1 from the base case with $\mu \in R_k$ and at_{k+1} respectively.

□

Chapter 7

Evaluation

In this chapter we first present the architecture of the implemented system. Since entailment regimes only change the evaluation of basic graph patterns, standard SPARQL algebra processors can be used that allow for custom BGP evaluation. Furthermore, standard OWL reasoners can be used to perform the required reasoning tasks.

7.1 The System Architecture

Figure 8.1 depicts the main phases of query processing in our prototypical system. In our setting, the queried graph is seen as an ontology that is loaded into an OWL reasoner. Loading the ontology and the initialization of the reasoner are performed before the system accepts queries. We use the ARQ library[1] of the Jena Semantic Web Toolkit for parsing the query and for the SPARQL algebra operations apart from our custom BGP evaluation method. The BGP is parsed and mapped into axiom templates by our BGP parser. The resulting axiom templates are then passed to a query optimizer, which applies the axiom template rewriting and then searches for a good query execution plan based on statistics provided by the reasoner. Afterwards, the query is evaluated as described in Algorithm 4 of Section 6. For the evaluation in the next sections we use the HermiT reasoner[2] for OWL reasoning, but any other reasoner that implements the standard reasoning tasks could be used.

7.2 Experimental Results

We tested the developed optimizations with standard benchmarks and a range of custom queries that test complex axiom template evaluation over more expressive ontologies. All experiments were performed on a Mac OS X Lion machine with a 2.53 GHz Intel Core i7 processor and Java 1.6 allowing 1GB of Java heap space. We measure the time for one-off tasks such as classification separately since such tasks are usually performed before the system accepts queries. The ontologies and all code required to perform the experiments are available online.[3] The developed system,[4]

[1] `http://jena.sourceforge.net/ARQ/`
[2] `http://www.hermit-reasoner.com/`
[3] `http://code.google.com/p/query-ordering/`
[4] `http://code.google.com/p/owl-bgp/`

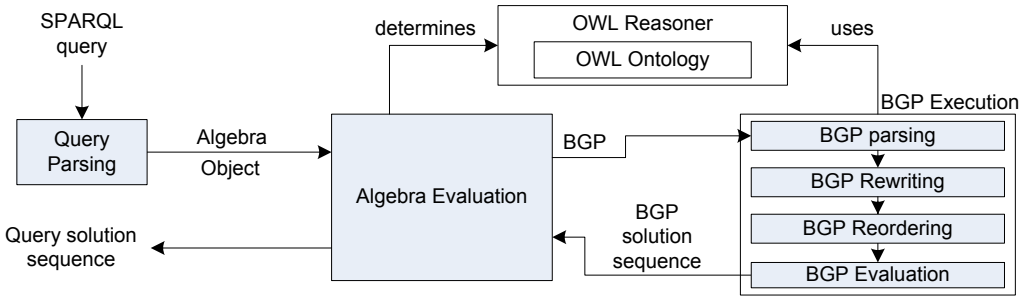

Figure 7.1: *The main phases of query processing in our system*

called OWL-BGP, is implemented as a SPARQL wrapper that can be used with any reasoner that implements the OWLReasoner interface of the OWL API [55]. In Section 7.2.1 we compare the different ordering strategies that have been developed on two benchmarks (LUBM and UOBM) that contain queries with variables only in place of individuals (i.e., queries containing only query atoms). We also show the effect of ordering on LUBM using some custom queries with simple axiom templates created for SPARQL-DL [84]. In Section 7.2.2 we show the effect of the proposed optimizations for queries with complex axiom templates. For the evaluation we have used the HermiT hypertableau reasoner.[5] Other reasoners such as Pellet[6] or Racer Pro[7] could equally well be used with our implementation as long as they provide an interface with the required ordering statistics, i.e., the number of known and possible instances of concepts and roles for the computation of the cost functions used for query ordering. Without any optimizations, providing this interface with statistics can easily be realized as described in Chapter 4. The presented query ordering techniques can also be used when optimizations such as caching, pseudo model merging techniques, binary instance retrieval, or absorption are employed. The cost functions might, however, require some adaptation to take the reduction in the required number of consistency checks into account as discussed in Section 4.6.

It is worth noting that the TrOWL reasoning framework[8] started to use our SPARQL wrapper to provide SPARQL support. An adaptation to also provide statistics is, to the best of our knowledge, still outstanding, although this should be straightforward. As it has been discussed in Section 3.2.4, TrOWL is based on two approximate reasoners: one that under-approximates (computation of concept and role instances is sound, but incomplete) [102] and one that over-approximates (computation of concept and role instances is complete, but unsound) [103]. In such a setting, the under-approximation can straightforwardly be seen as the known instances and the over-approximation minus the under-approximation as the possible instances.

7.2.1 Query Ordering

We tested our ordering techniques with the Lehigh University Benchmark (LUBM) [45] as a case where no disjunctive information is present and with the more expressive University Ontology Benchmark (UOBM) [88].

[5] `http://www.hermit-reasoner.com/`
[6] `http://clarkparsia.com/pellet/`
[7] `http://www.racer-systems.com`
[8] `http://trowl.eu`

We first used the 14 conjunctive ABox queries provided in LUBM. From these, Queries 2, 7, 8, 9 are the most interesting ones in our setting since they contain many atoms and ordering them can have an effect in running time. We tested the queries on LUBM(1,0) and LUBM(2,0) which contain data for one and two universities respectively, starting from index 0. LUBM(1,0) contains 17,174 individuals and LUBM(2,0) contains 38,334 individuals. LUBM(1,0) took 19 s to load and 0.092 s for classification and initialization of known and possible instances of concepts and roles and known and possible equivalent individuals. The clustering approach for concepts took 1 s and resulted in 16 clusters. The clustering approach for roles lasted 4.9 s and resulted in 17 role successor clusters, 29 role predecessor clusters and 87 role clusters. LUBM(2,0) took 48.5 s to load and 0.136 s for classification and initialization of known and possible instances. The clustering approach for concepts took 3.4 s and resulted in 16 clusters. The clustering approach for roles lasted 16.3 s and resulted in 17 role successor clusters, 31 role predecessor clusters and 102 role clusters. Table 7.1 shows the execution time for each of the four queries for LUBM(1,0) and LUBM(2,0) for four cases: i) when we use the static algorithm (columns 2 and 6), ii) when we use the dynamic algorithm (columns 3 and 7), iii) when we use random sampling, i.e., taking half of the individuals that are returned (from the evaluation of previous query atoms) in each run, to decide about the next cheapest atom to be evaluated in the dynamic case (columns 4 and 8) and iv) using the proposed sampling approach that is based on clusters constructed from individuals in the queried ontology (columns 5 and 9). The queries marked with (*) are the queries where the static and dynamic algorithms result in a different ordering. In Queries 7 and 8 we observe an increase in running time when the dynamic technique is used (in comparison to the static) which is especially evident on Query 8 of LUBM(2,0), where the number of individuals in the ontology and the intermediate result sizes are larger. Dynamic ordering also behaves worse than static in Queries 2 and 9. This happens because, although the dynamic algorithm chooses a better ordering than the static algorithm, the intermediate results (that need to be checked in each iteration to determine the next query atom to be executed) are quite large and hence the cost of iterating over all possible mappings in the dynamic case far outweighs the better ordering that is obtained. We also observe that a random sampling for collecting the ordering statistics in the dynamic case (checking only 50% of individuals in Ω_{i-1} randomly for detecting the next query atom to be executed) leads to much worse results in most queries than plain static or dynamic ordering. This happens since random sampling often leads to the choice of a worse execution order. The use of the cluster based sampling method performs better than the plain dynamic algorithm in all queries. In Queries 2 and 9, the gain we have from the better ordering of the dynamic algorithm when sampling is used is much more evident. This is the case since we use at most one individual from every cluster for the cost functions computation and the number of clusters is much smaller than the number of the otherwise tested individuals in each run.

In order to show the effectiveness of our proposed cost functions we compared the running times of all the valid plans (plans that comply to the connectedness condition of Definition 12 of Section 4.3, i.e., plans on which consecutive atoms share at least one common variable) with the running time of the plan chosen by our method. In the following we show the results for LUBM(1, 0), but the results for LUBM(2,0) are comparable. In Table 7.2 we show, for each query, the number

Table 7.1: *Query answering times in milliseconds for LUBM(1,0) and LUBM(2,0) using i) the static algorithm ii) the dynamic algorithm, iii) 50% random sampling (RSampling), iv) the constructed individual clusters for sampling (CSampling)*

	LUBM(1,0)				LUBM(2,0)			
Q	Static	Dynamic	RSampling	CSampling	Static	Dynamic	RSampling	CSampling
∗2	51	119	390	37	162	442	1,036	153
7	25	29	852	20	70	77	2,733	64
8	485	644	639	551	622	866	631	660
∗9	1,099	2,935	3,021	769	6,108	23,202	14,362	3,018

Table 7.2: *Statistics about the constructed plans and chosen orderings and running times in milliseconds for the orderings chosen by Pellet and for the worst constructed plans*

Q	PlansNo	Chosen Plan Order			Pellet Plan	Worst Plan
		Static	Dynamic	Sampling		
2	336	2	1	1	51	4,930
7	14	1	1	1	25	7,519
8	56	1	1	1	495	1,782
9	336	173	160	160	1,235	5,388

of valid plans (column 2), the order of the plan chosen by the static, dynamic, and cluster based sampling methods if we order the valid plans by their execution time (columns 3,4,5; e.g., a value of 2 indicates that the ordering method chose the second best plan), the running time of HermiT for the plan that was created by Pellet (column 6) as well as the running time of the worst constructed plan (column 7).

The comparison of our ordering approach with the approach followed by other reasoners that support conjunctive query answering such as Pellet or Racer Pro is not very straightforward. This is the case because Pellet and Racer have many optimizations for instance retrieval [126, 48], which HermiT does not have. Thus, a comparison between the execution times of these reasoners and HermiT would not convey much information about the effectiveness of the proposed query ordering techniques. The idea of comparing only the orderings computed by other reasoners with those computed by our methods is also not very informative since the orderings chosen by different reasoners depend on the costs of specific tasks in these reasoners and, hence, are, to some extent, reasoner dependent, i.e., an ordering that is good for one reasoner and which leads to an efficient evaluation may not be good for another reasoner. We should note that when we were searching for orderings according to Pellet, we switched off the simplification optimization that Pellet implements regarding the exploitation of domain and range axioms of the queried ontology for reducing the number of query atoms to be evaluated [124]. This has been done in order to better evaluate the difference of the plain ordering obtained by Pellet and HermiT since our cost functions take into account all the query atoms.

We observe that for all queries apart from Query 9 the orderings chosen by our algorithms are the (near)optimal ones. For Queries 2 and 7, Pellet chooses the same ordering as our algorithms. For Query 8, Pellet chooses an ordering which, when evaluated with HermiT, results in higher execution time. For Query 9, our algorithms choose plans from about the middle of the order over all the valid plans w.r.t. the query execution time, which means that our algorithms do not perform well in this query. This is because of the greedy techniques we have used to find the execution plan which take into account only local information to choose the next query atom to be executed. The ordering chosen by Pellet for Query 9 does also not

Table 7.3: *Number of individuals in LUBM with increasing number of universities*

LUBM(3,0)	LUBM(4,0)	LUBM(5,0)	LUBM(6,0)	LUBM(7,0)	LUBM(8,0)	LUBM(9,0)
55,664	78,579	102,368	118,500	144,612	163,552	183,425

Table 7.4: *Query answering times in seconds for LUBM with increasing number of universities*

Q	LUBM(3,0)	LUBM(4,0)	LUBM(5,0)	LUBM(6,0)	LUBM(7,0)	LUBM(8,0)	LUBM(9,0)
2	0.35	0.62	1.26	1.71	2.26	3.11	4.18
7	0.11	0.16	0.23	0.32	0.33	0.33	0.40
8	0.77	0.91	1.27	1.29	1.34	1.44	1.65
9	18.49	42.98	85.54	116.88	181.07	235.06	312.71
all	20.64	55.16	90.99	138.84	213.59	241.85	323.15

perform well. We see that, in all queries, the worst running times are many orders of magnitude greater than the running times achieved by our ordering algorithms. In general, we observe that in LUBM static techniques are adequate and the use of dynamic ordering does not improve the execution time compared to static ordering.

In order to show the scalability of the system, we next run the LUBM queries with different numbers of universities, i.e., LUBM(i,0) where i ranges from 3 to 9. Table 7.3 shows the number of individuals appearing in each ABox of different university size. The running times of Queries 2, 7, 8, 9 as well as the running time of all the 14 LUBM queries are shown in Table 7.4. Note that the results shown are for the case that static ordering is performed. From this table we see that for all queries, the running time increases when the number of individuals of the ABox increases, which is reasonable. We observe that query answering over ontologies is still not as scalable as query answering over databases and this is so, because of the more expressive schema that has to be taken into account and the fact that we have incomplete information in contrast to databases where we have complete information.

Unlike LUBM, the UOBM ontology contains disjunctions and the reasoner makes also non-deterministic derivations. In order to reduce the reasoning time, we removed the nominals and only used the first three departments containing 6,409 individuals. The resulting ontology took 16 s to load and 0.1 s to classify and initialize the known and possible instances. The clustering approach for concepts took 1.6 s and resulted in 356 clusters. The clustering approach for roles lasted 6.3 s and resulted in 451 role successor clusters, 390 role predecessor clusters and 4,270 role clusters. We present results for the static and dynamic algorithms on Queries 4, 9, 11, 12 and 14 provided in UOBM, which are the most interesting ones because they consist of many atoms. Most of these queries contain one atom with possible instances. As we see from Table 7.5, static and dynamic ordering show similar performance in Queries 4, 9, 11 and 12. Since the available statistics in this case are quite accurate, both methods find the optimal plans and the intermediate result set sizes are small. For both ordering methods, atoms with possible instances for these queries are executed last. In Query 14, the dynamic algorithm finds a better ordering which results in better performance. The effect that the cluster based sampling technique has on the running time is not as obvious as in the case of LUBM. This happens because in the current experiment the intermediate result sizes are not very large and, most importantly, because the gain obtained due to sampling is in the order of milliseconds whereas the total query answering times are in the order of

Table 7.5: *Query answering times in seconds for UOBM (1 university, 3 departments) and statistics*

Q	Static	Dynamic	CSampling	PlansNo	Chosen Plan Order			Pellet	Worst
					Static	Dynamic	Sampling	Plan	Plan
4	13.35	13.40	13.41	14	1	1	1	13.40	271.56
9	186.30	188.58	185.40	8	1	1	1	636.91	636.91
11	0.98	0.84	1.67	30	1	1	1	0.98	> 30 min
12	0.01	0.01	0.01	4	1	1	1	0.01	> 30 min
14	94.61	90.60	93.40	14	2	1	1	> 30 min	> 30 min
q_1	191.07	98.24	100.25	6	2	1	1	> 30 min	> 30 min
q_2	47.04	22.20	22.51	6	2	1	1	22.20	> 30 min

seconds obscuring the small improvement in running time due to sampling. In all queries the orderings that are created by Pellet result in the same or worse running times than the orderings created by our algorithms.

In order to illustrate when dynamic ordering performs better than static, we also created the two custom queries:

$$q_1 = \{ \mathsf{isAdvisedBy}(?x,?y), \mathsf{GraduateStudent}(?x), \mathsf{Woman}(?y) \}$$
$$q_2 = \{ \mathsf{SportsFan}(?x), \mathsf{GraduateStudent}(?x), \mathsf{Woman}(?x) \}$$

In both queries, $P[\mathsf{GraduateStudent}]$, $P[\mathsf{Woman}]$ and $P[\mathsf{isAdvisedBy}]$ are non-empty, i.e., the query concepts and roles have possible instances. The running times for dynamic ordering are smaller since the more accurate statistics result in a smaller number of possible instances that have to be checked during query execution. In particular, for the static ordering, 151 and 41 possible instances have to be checked in query q_1 and q_2, respectively, compared to only 77 and 23 for the dynamic ordering. Moreover, the intermediate results are generally smaller in dynamic ordering than in static leading to a significant reduction in the running time of the queries. Interestingly, query q_2 could not be answered within the time limit of 30 minutes when we transformed the three query concepts into a conjunction, i.e., when we asked for instances of the intersection of the three concepts. This is because for complex concepts the reasoner can no longer use the information about known and possible instances and falls back to a more naive way of computing the concept instances. Again, for the same reasons as before, the sampling techniques have no apparent effect on the running time of these queries.

For each query of the SPARQL-DL tests issued over LUBM(1,0) [84] (see Table 7.6), Table 7.7 shows the running time of the plan chosen by our method (column 2), the number of valid plans, i.e., plans that comply to the connectedness condition of Definition 12 (column 3), the order of the chosen plan if we order the valid plans by their execution times (column 4), the running time of HermiT for the plan that was created by Pellet (column 5) as well as the running time of the worst constructed plan (column 6). The queries as shown in Table 7.6 are ordered according to our static ordering algorithm. Since reasoning for LUBM is deterministic, we use static planning to order the axiom templates. Dynamic planning does not improve the execution times (actually it makes them worse) since, as it has been explained before, with only deterministic reasoning we have most of the important information for ordering from the beginning and the overhead caused by dynamic ordering results in worse query execution time.

From the results of Table 7.7 one can observe that for Queries 1, 2, 3, 4 and 8 the proposed ordering chooses the optimal plan among all valid plans. For Queries 5, 6, 7,

Table 7.6: *Queries used for SPARQL-DL tests*

1	$\mathsf{GraduateStudent}(?x)$ $?y(?x, ?z)$ $\mathsf{Course}(?w)$	6	$\mathsf{GraduateStudent}(?x)$ $?y(?x, ?w)$ $?z(?w)$ $\mathsf{GraduateCourse} \sqsubseteq \neg ?z$
2	$?c \sqsubseteq \mathsf{Employee}$ $?c(?x)$ $\mathsf{Student}(?x)$ $\mathsf{undergraduateDegreeFrom}(?x, ?y)$	7	$?c \sqsubseteq \top$ $?c(?x)$ $\mathsf{teachingAssistantOf}(?x, ?y)$ $\mathsf{takesCourse}(?x, ?y)$
3	$?y \sqsubseteq \mathsf{memberOf}$ $?y(?x, \mathrm{University0})$ $\mathsf{Person}(?x)$	8	$?c \sqsubseteq \mathsf{Person}$ $?c(?x)$ $\mathsf{advisor}(?x, ?y)$
4	$?y(\mathsf{GraduateStudent5}, ?w)$ $?z(?w)$ $?z \sqsubseteq \mathsf{Course}$	9	$?c \sqsubseteq \mathsf{Person}$ $?c(?x)$ $\mathsf{teachingAssistantOf}(?x, ?y)$ $\mathsf{Course}(?y)$
5	$?z \sqsubseteq \mathsf{Course}$ $?z(?w)$ $?y(?x, ?w)$ $\mathsf{GraduateStudent}(?x)$	10	$?p \sqsubseteq \mathsf{worksFor}$ $?p(?y, ?w)$ $?c(?y)$ $?c \sqsubseteq \mathsf{Faculty}$ $\mathsf{advisor}(?x, ?y)$ $\mathsf{GraduateStudent}(?x)$ $\mathsf{memberOf}(?x, ?w)$

Table 7.7: *Query answering times in seconds for the queries of Table 7.6 over LUBM(1,0) and statistics*

Query	Chosen Ordering Time	PlansNo	Chosen Plan Order	Pellet Plan Time	Worst Plan Time
1	0.36	2	1	0.81	0.81
2	0.03	14	1	0.03	0.61
3	0.05	4	1	0.05	5.45
4	0.01	4	1	0.01	11.46
5	26.10	8	5	3.30	454.25
6	10.49	8	2	11.90	499.65
7	0.42	14	6	0.42	2.68
8	0.23	4	1	0.23	0.80
9	0.19	8	4	0.19	0.47
10	0.80	812	21	0.93	992.77

9 and 10 the optimal plan is not chosen according to the proposed cost estimation algorithm. For Queries 5, 7, 9 and 10 this is due to the greedy techniques we have used for finding in each iteration of our ordering algorithm the next cheapest axiom template to be evaluated. For example, the optimal plan for Query 10 starts with the template $\mathsf{GraduateStudent}(?x)$, which is not the cheapest one according to our cost based technique and then, while moving over connected templates, a different order is chosen than the order chosen by our algorithm. It turns out that all valid plans beginning with the atom $\mathsf{GraduateStudent}(?x)$ lead to better execution times than the plan chosen by our algorithm resulting in the existence of several better plans than the chosen one. For Query 6 we do not find the optimal plan because we have overestimated the cost of the disjoint axiom template and hence have missed

the optimal ordering.

Nevertheless, the chosen plans lead to execution times for all queries that are up to three orders of magnitude lower than those when the worst plans are chosen. For queries in which the proposed ordering does not lead to the optimal plan, one has to additionally take into account the time we saved from not computing the costs for the $|q!|$ possible orderings, which can be very high. We observe that for all queries apart from Query 5, the orderings created by Pellet result in the same or worse running times than the orderings created by our algorithms. Interestingly for Query 5, Pellet finds a plan that when evaluated with HermiT results in better execution time than the plan chosen by our algorithms. As discussed before, HermiT misses the optimal plan in this case because of the greedy techniques that we use for query ordering according to which HermiT chooses first the locally cheapest template $?z \sqsubseteq \mathsf{Course}$, whereas the plan chosen by Pellet starts with the template $?y(?x, ?w)$, which results in a better ordering.

7.2.2 Complex Axiom Template Optimizations

In the absence of suitable standard benchmarks for arbitrary SPARQL queries, we created a custom set of queries as shown in Tables 7.9 and 7.11 for the GALEN and the FBbt_XP ontology, respectively. Systems that fully support the SPARQL Direct Semantics entailment regime are still under development, which makes it hard to compare our results for these kinds of queries with other systems.

GALEN is a biomedical ontology. It's expressivity is (Horn-)$\mathcal{SHIF}$ and it consists of 2,748 concepts and 413 abstract roles. FBbt_XP is an ontology taken from the Open Biological Ontologies (OBO) Foundry [127]. It falls into the $\mathcal{SHI}$ fragment of $\mathcal{SROIQ}$ and consists of 7,221 concepts and 21 abstract roles. We only consider the TBox part of FBbt_XP since the ABox is not relevant for showing the different effects of the proposed optimizations on the execution times of the considered queries. GALEN took 3.7 s to load and 11.1 s to classify (concepts and roles), while FBbt_XP took 1.5 s to load and 7.4 s to classify.

The execution times for the queries of Tables 7.9 and 7.11 are shown on the right-hand side of Tables 7.8 and 7.10, respectively. We have set a time limit of 30 minutes for each query. For each query, we tested the execution once without optimizations and once for each combination of applicable optimizations from Sections 4.3 and 5. In Tables 7.8 and 7.10, one can also see the number of consistency checks that were performed for the evaluation of each query and each combination of the applicable optimizations as well as the number of results of each query. In these tables we have taken the time of the worst ordering of query atoms for the cases in which the ordering optimization is applicable but not enabled. Note that only the complex axiom templates require consistency checks to be evaluated; the simple ones (subsumption axiom templates in this case) need only cache look-ups in the reasoner's internal structures since the concepts and roles are already classified.

GALEN Queries: As expected, an increase in the number of variables within an axiom template leads to a significant increase in the query execution time because the number of mappings that have to be checked grows exponentially in the number of variables. This can, in particular, be observed from the difference in execution time between Query 1 and 2. From these two queries, it is evident that the use of the hierarchy exploitation optimization leads to a decrease in execution time of up

Table 7.8: *Query answering times in seconds for the queries of Table 7.9 with and without optimizations*

Query	Ordering	Hierarchy Exploitation	Rewriting	Consistency Checks	Time	AnswersNo
1				2,750	1.68	10
1		x		50	0.18	10
2				1,141,250	578.98	214
2		x		1,291	9.85	214
3		x			>30 min	
3			x	19,250	102.37	2,816
3		x	x	3,073	2.69	2,816
4	x	x			> 30 min	
4		x	x		> 30 min	
4	x		x	16,135	7.68	51
4	x	x	x	197	1.12	51
5		x			> 30 min	
5	x			1,883	0.67	4,392
5	x	x		1,883	0.8	4,392

Table 7.9: *Sample complex queries for the GALEN ontology*

1	$\mathsf{Infection} \sqsubseteq \exists \mathsf{hasCausalLinkTo}.?x$
2	$\mathsf{Infection} \sqsubseteq \exists ?y.?x$
3	$?x \sqsubseteq \mathsf{Infection} \sqcap \exists \mathsf{hasCausalAgent}.?y$
4	$\mathsf{NAMEDLigament} \sqsubseteq \mathsf{NAMEDInternalBodyPart} \sqcap ?x$
	$?x \sqsubseteq \exists \mathsf{hasShapeAnalagousTo}?y \sqcap \exists ?z.\mathsf{linear}$
5	$?x \sqsubseteq \mathsf{NonNormalCondition}$
	$?z \sqsubseteq \mathsf{ModifierAttribute}$
	$\mathsf{Bacterium} \sqsubseteq \exists ?z.?w$
	$?y \sqsubseteq \mathsf{StatusAttribute}$
	$?w \sqsubseteq \mathsf{AbstractStatus}$
	$?x \sqsubseteq \exists ?y.\mathsf{Status}$

to two orders of magnitude. Query 3 can only be completed in the time limit if at least the query rewriting optimization is enabled. We can get an improvement of up to three orders of magnitude in this query, by using rewriting in combination with the hierarchy exploitation. Query 4 can only be completed in the given time limit if at least ordering and rewriting is enabled. Rewriting splits the first axiom template into the following two simple axiom templates, which are evaluated much more efficiently:

$$\mathsf{NAMEDLigament} \sqsubseteq \mathsf{NAMEDInternalBodyPart} \quad \text{and} \quad \mathsf{NAMEDLigament} \sqsubseteq ?x$$

After the rewriting, the ordering optimization has an even more pronounced effect since both rewritten axiom templates can be evaluated with a simple cache look-up. Without ordering, the complex axiom template could be executed before the simple ones, which leads to the inability of answering the query within the time limit of 30 min. Without a good ordering, Query 5 can also not be answered within the time limit. The ordering chosen by our algorithm is shown below. Note that the query consists of two connected components: one for the axioms containing $?z$ and $?w$ and

another one for the axioms containing $?x$ and $?y$.

$$\begin{aligned} ?z &\sqsubseteq \mathsf{ModifierAttribute} \\ ?w &\sqsubseteq \mathsf{AbstractStatus} \\ \mathsf{Bacterium} &\sqsubseteq \exists ?z.?w \end{aligned}$$

$$\begin{aligned} ?y &\sqsubseteq \mathsf{StatusAttribute} \\ ?x &\sqsubseteq \mathsf{NonNormalCondition} \\ ?x &\sqsubseteq \exists ?y.\mathsf{Status} \end{aligned}$$

In this query, the hierarchy exploitation optimization does not improve the execution time since, due to the chosen ordering, the variables on which the hierarchy optimization can be applied, are already bound when it comes to the evaluation of the complex templates. Hence, the running times with and without the hierarchy exploitation are similar. The number of consistency checks is significantly lower than the number of answers because the overall results are computed by taking the cartesian products of the results for the two connected components. Interestingly, for queries with complex axiom templates, it does not make sense to require that the next axiom template to evaluate shares a variable with the previously evaluated axiom templates, as in the case of simple axiom templates. For example, if we would require that, the first connected component of the query would be executed in the following order:

$$\begin{aligned} ?z &\sqsubseteq \mathsf{ModifierAttribute} \\ \mathsf{Bacterium} &\sqsubseteq \exists ?z.?w \\ ?w &\sqsubseteq \mathsf{AbstractStatus} \end{aligned}$$

this results in 294,250 instead of 1,498 consistency checks since we no longer use a cheap cache look-up check to determine the bindings for $?w$, but first iterate over all possible $?w$ bindings and check entailment of the complex axiom template and then reduce the computed candidates when processing the last axiom template.

Although our optimizations can significantly improve the query execution time, the required time can still be quite high. In practice, it is, therefore, advisable to add as many restrictive axiom templates (axiom templates which require only cache look-ups) for query variables as possible. For example, the addition of $?y \sqsubseteq \mathsf{Shape}$ to Query 4 reduces the runtime from 1.12 s to 0.65 s. We observe, as expected, that the execution time for each query and applicable optimization is analogous to the number of consistency checks that are performed for the evaluation of the query.

FBbt_XP Queries: For Queries 1, 2, 3, 5 and 6, on which the ordering optimization is applicable, we observe a decrease in execution time up to two orders of magnitude when the ordering optimization is used. The ordering optimization is important for answering Queries 1, 2 and 3 within the time limit. For all queries, the additional use of the hierarchy exploitation optimization leads to an improvement of up to three orders of magnitude. We observe that in some queries the effect of the hierarchy exploitation is more profound than in others. More precisely, the smaller the ratio of the result size to the number of consistency checks without the hierarchy optimization, the more pronounced is the effect when enabling this optimization. In other words, when more tested mappings are indeed solutions, one can prune fewer parts of the hierarchy since pruning can only be performed when

Table 7.10: *Query answering times in seconds for the queries of Table 7.11 with and without optimizations*

Query	Ordering	Hierarchy Exploitation	Rewriting	Consistency Checks	Time	AnswersNo
1	x			151,683	44.13	7,243
1		x			> 30 min	
1	x	x		11,262	5.64	7,243
2	x			14,446	37.38	7,224
2		x			> 30 min	
2	x	x		12,637	39.20	7,224
3	x			72,230	357.59	188
3		x			> 30 min	
3	x	x		54,186	252.41	188
4				166,129	486.81	68
4		x		1335	17.03	68
5				166,129	457.84	0
5	x			21,669	19.68	0
5		x		907	11.74	0
5	x	x		3	0.01	0
6	x	x			> 30 min	
6	x		x	43,338	183.66	43,338
6		x	x		> 30 min	
6	x	x	x	32,490	152.38	43,338

Table 7.11: *Sample complex queries for the FBbt_XP ontology*

1	$?x \sqsubseteq \forall \mathsf{part\_of}.?y$ $?x \sqsubseteq \mathsf{FBbt\_00005789}$	4	$\mathsf{FBbt\_00001606} \sqsubseteq \exists ?y.?x$
2	$?y \sqsubseteq \mathsf{part\_of}$ $?x \sqsubseteq \forall ?y.\mathsf{FBbt\_00001606}$	5	$\mathsf{FBbt\_00001606} \sqsubseteq \exists ?y.?x$ $?y \sqsubseteq \mathsf{develops\_from}$
3	$?x \sqsubseteq \exists ?y.\mathsf{FBbt\_00025990}$ $?y \sqsubseteq \mathsf{overlaps}$	6	$?y \sqsubseteq \mathsf{FBbt\_00001884}$ $?p \sqsubseteq \mathsf{part\_of}$ $?x \sqsubseteq \exists ?p.?y \sqcap ?w$

we find a non-solution. In Query 2, we even observe a slight increase in running time when the hierarchy optimization is used. This is because the optimization can only prune few candidate mappings, which does not outweigh the overhead caused by maintaining information about which hierarchy parts have already been tested. In Query 6, the rewriting optimization is important to answer the query within the time limit. When all optimizations are enabled, the number of consistency checks is less than the result size (32,490 versus 43,338) since only the complex axiom template requires consistency checks.

Chapter 8

Semantic Access to Cultural Content

The current state of the art in cultural heritage implements a model whereby many aggregators, content providers and projects feed their content into a national, thematic, or European portal, and this portal is then used by the end user to find cultural items. Typically, the content is described with the aid of standard schemas (metadata schemas) that contain information about resources. Europeana[1] is being developed as the single, direct and multilingual gateway to Europe's cultural heritage that provides integrated access to digital objects from cultural heritage organizations (museums, libraries, archives and audio-visual archives). Several cross-domain, vertical or thematic aggregators are being deployed at regional, national and international level in order to reinforce this initiative by collecting and converting metadata about existing and newly digitized resources. Due to the diversity of content types and of metadata schemas used to annotate the content, semantic interoperability has been identified as a key issue during the last years.[2]

The main approach to interoperability of cultural content metadata has been the usage of well-known standards in the specific museum, archive and library sectors and their mapping to a common data model used at the European level. The currently employed Europeana Semantic Elements[3] (ESE) Model is a Dublin Core[4]-based application profile providing a generic set of terms that can be applied to heterogeneous materials thereby providing a baseline to allow contributors to take advantage of their existing rich descriptions. The latter constitute a knowledge base that is constantly growing and evolving, both by newly introduced annotations and digitization initiatives, as well as through the increased efforts and successful outcomes of the aggregators and the content providing organizations.

The new Europeana Data Model[5] (EDM) is introduced as a data structure aiming to enable the linking of data and to connect and enrich descriptions in accordance with the Semantic Web developments. Its scope and main strength is the adoption of an open, cross-domain framework in order to accommodate the growing number of rich, community-oriented standards such as LIDO (Lightweight Information De-

[1] `http://www.europeana.eu`
[2] First Workshop on Semantic Interoperability in the European Digital Library (SIEDL 2008)
[3] `http://www.europeana.eu/schemas/ese/`
[4] `http://dublincore.org/`
[5] `http://www.europeana.eu/schemas/edm/`

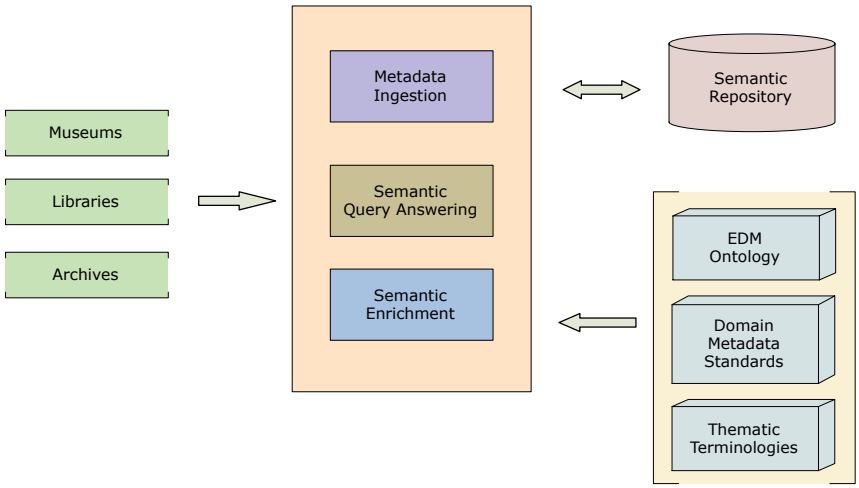

Figure 8.1: *The architecture of the proposed metadata aggregation and semantic query answering system*

scribing Objects)[6] for museums, EAD (Encoded Archival Description)[7] for archives or METS (Metadata Encoding and Transmission Standard)[8] for libraries. Apart from its ability to support standards of high richness, EDM also enables source aggregation and data enrichment from a range of third party sources while clearly providing the provenance of all information.

Following ongoing efforts to investigate usage of the semantic layer as a means to improve user experience, we are facing the need to provide a more detailed semantic description of cultural content. Semantic description of cultural content, accessible through its metadata, would be of little use, if users were not in position to pose their queries in terms of a rich integrated ontological knowledge. Currently, this is performed through a data storage schema, which highly limits the aim of the query. Therefore the use of query answering techniques that are based not only on string matching over data that are stored in databases, but also on the implicit meaning that can be found by reasoning based on detailed *domain terminological knowledge* is important. In this way, content metadata can be terminologically described, semantically connected and used in conjunction with other, useful, possibly complementary content and information, independently published on the web. A semantically integrated cultural heritage knowledge, facilitating access to cultural content is, therefore, achieved. The key for this is to semantically connect metadata with ontological domain knowledge through appropriate mappings.

Figure 8.1 shows the architecture of the whole system for access to cultural heritage content, depicting the workflow of cultural aggregation processing and query answering. On the left hand side, cultural content providers (museums, libraries, archives) and aggregators wish to make their content visible to Europeana. This is performed by ingesting (usually a subset of) their content metadata descriptions to the Europeana portal. This is a rather difficult task, mainly due to the heterogeneity of the metadata storage schemas (from both technological and conceptual point of

[6] `http://www.lido-schema.org`
[7] `http://www.loc.gov/ead/`
[8] `http://www.loc.gov/standards/mets/`

view) that need to be transformed to the EDM form. The *Metadata Ingestion* module provides users with the ability to map and transform their data to EDM elements through a graphical interface and an associated automatic procedure. The result of this module is an EDM version of the cultural content metadata. Moreover, through the *Semantic Enrichment* module, the translated metadata are represented as RDF triples, in the form of formal assertional knowledge and the Semantic Web principles, and stored in the *Semantic Repository*.

The metadata elements are represented in the semantic repository as descriptions of individuals, i.e., connections of individuals with entities of the *terminological knowledge*. This knowledge is an ontological representation of EDM (the *EDM Ontology*), that is connected, on the one hand, to *Domain Metadata Standards* (Dublin Core, LIDO, CIDOC CRM[9] etc) sharing terminology with them and providing the general description of 'Who?', 'What?', 'When?' and 'Where?' for every digital object and, on the other hand, to more specific terminological axioms providing details about species, categories, properties, interrelations etc. (e.g., brooches are made of copper or gold). The latter knowledge (the *Thematic Ontologies*) is developed by the providers and aggregators and can be used both for semantic enrichment of content metadata, and for reasoning in the *Semantic Query Answering* module. Thus, it provides the user with the ability to build complex queries in terms of the above terminology and access cultural content effectively.

In this chapter, we describe a cultural heritage application of the ontological query answering techniques that have been described in the previous chapters. In the next section, we first briefly describe how the enrichment of metadata is performed through the use of thematic ontologies. Afterwards, we focus on the query answering part of the system and show how we can combine the already implemented system described in Chapter 6 with a query rewriting system to efficiently answer user queries. In Section 8.2 we show an experimental study of the proposed system.

8.1 Metadata Enrichment and Query Answering for Improved Resource Discovery

As we already described, the metadata are initially collected from data providers and mapped to a common semantic data model, i.e., EDM in our case. The transformation of the data of content providers to data in terms of the EDM ontology results in a set of RDF triples that are more like an attribute-value set for each object. An example of an RDF instance record is shown in Figure 8.2 where one can see that EDM, on the one hand, re-uses terms from other namespaces, like RDF(S), SKOS (Simple Knowledge Organization System) or DC (Dublin Core) namespaces and, on the other hand, introduces new terms. Since the EDM ontology is a general ontology referring to metadata descriptions for each object, the usage of thematic ontologies for different domains is necessary in order to add semantically processable information to each object. For example, the information that an object is of type vase may not be adequate for a specific application; one may be interested in the specific type of vase, like amphora or alabaster, or, in absence of such information, in the characteristics that a vase should have in order to be classified to a specific type. First, thematic ontologies are created in collaboration with field experts.

[9] www.cidoc-crm.org/

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:baseURI="http://baseURI/"
 xmlns:skos="http://www.w3.org/2004/02/skos/core#"
 xmlns:ens="http://www.europeana.eu/schemas/edm/"
 xmlns:ore="http://www.openarchives.org/ore/terms/"
 xmlns:dcterms="http://purl.org/dc/terms/"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:ese="http://www.europeana.eu/schemas/ese/" >
<rdf:Description rdf:about="http://baseURI/PhysicalThing/Local ΙΔΕΠ/ΜΒΠ/54/28213">
 <rdf:type>Museum object</rdf:type>
 <rdf:type rdf:resource="http://www.europeana.eu/schemas/edm/PhysicalThing"/>
</rdf:Description>
<rdf:Description rdf:about="http://baseURI/Aggregation/AggregationRes139">
 <ens:landingPage
rdf:resource="http://collections.culture.gr/ItemPage.aspx?ObjectID=1933"/>
 <dc:creator>Athena; Greece</dc:creator>
 <ens:aggregatedCHO rdf:resource="http://baseURI/PhysicalThing/Local
ΙΔΕΠ/ΜΒΠ/54/28213"/>
 <rdf:type rdf:resource="http://www.openarchives.org/ore/terms/Aggregation"/>
</rdf:Description>
<rdf:Description rdf:about="http://baseURI/Proxy/ProxyRes139">
 <dcterms:spatial></dcterms:spatial>
 <dc:type>Επιγραφή Ατομική</dc:type>
 <dc:title>Επιγραφή Ιδιωτική</dc:title>
 <dc:source>Υπουργείο Πολιτισμού - Τουρισμού</dc:source>
 <dc:identifier>ΕΠ/ΜΒΠ/54/28</dc:identifier>
 <ens:language>Greek</ens:language>
 <ens:proxyIn rdf:resource="http://baseURI/Aggregation/AggregationRes139"/>
 <dc:type>Επιγραφή επιτύμβια</dc:type>
 <dc:description>Επιγραφή. Πλάκα από φαιόλευκο μάρμαρο. Λείπει τμήμα της άνω
αριστερής γωνίας. Ύψος 20 εκ., πλάτος 14,2 εκ., πάχος 2,6 εκ., ύψος γραμμάτων 2-2,2 εκ.
Προέλευση: Θεσσαλονίκη, Παρεκκλήσι Πύργου Ανατολικού Τείχους, κοντά στο Τριγώνιο.
Κείμενο επιγραφής: ΥΠ(ΕΡ) / ΕΥΧΗΣ / Φ(Ι)Λ(ΙΠ)ΠΟΥ.</dc:description>
 <dcterms:created>5ος αιώνας</dcterms:created>
 <dc:rights>Υπουργείο Πολιτισμού - Τουρισμού</dc:rights>
 <dc:rights>Hellenic Ministry of Culture - Tourism</dc:rights>
 <dcterms:medium></dcterms:medium>
 <dcterms:spatial>Μουσείο Βυζαντινού Πολιτισμού</dcterms:spatial>
 <ens:country>Greece</ens:country>
 <dc:source>Hellenic Ministry of Culture - Tourism</dc:source>
 <rdf:type rdf:resource="http://www.openarchives.org/ore/terms/Proxy"/>
 <ens:provider>Athena, Greece</ens:provider>
 <ens:proxyFor rdf:resource="http://baseURI/PhysicalThing/Local ΙΔΕΠ/ΜΒΠ/54/28213"/>
 <ens:type>IMAGE</ens:type>
</rdf:Description>
</rdf:RDF
```

Figure 8.2: *RDF output of an example record*

These ontologies include individuals which represent objects, concepts which define sets of objects and roles defining relations between objects. Then, the data values filling the attributes of the EDM-RDF instances are transformed to individuals of the thematic ontologies. These individuals are then grouped together to form concepts as imposed by the thematic ontologies. The transformation of data values to individuals is performed, from a technical point of view, by mapping the data values to IRIs. After this transformation the data is stored in a semantic repository, from where they can be extracted through queries.

We now describe the techniques we have used to answer queries efficiently. In particular, we have used our implemented system and combined it with a query rewriting system. In this way, we exploit the advantages of both query answering methods.

The query rewriting method that we use works with the Description Logic DL-

Lite$_R$ [4, 18] (OWL 2 QL), which as stated in Section 2.1 is an extension of DL-Lite$_{core}$ that additionally allows for role hierarchies. As discussed in Section 3.2.2, the DL-Lite languages are appropriate for splitting the problem of query answering into two parts: the reasoning part which expands the initial query taking into account terminological knowledge provided by the TBox and the data retrieval part which retrieves the instances of the expanded query from the repository. We remind readers of the query rewriting approach through an example.

Example 11. *Let $\mathcal{T}$ be a TBox which consists of the two axioms :*

$$\mathsf{WorkOfArt} \sqsubseteq \exists \mathsf{madeBy}.\mathsf{Artist} \tag{8.1}$$

$$\mathsf{Painting} \sqsubseteq \mathsf{WorkOfArt} \tag{8.2}$$

and we ask the query

$$q = \{\mathsf{madeBy}(?x, ?y), \mathsf{Artist}(?y)\} \tag{8.3}$$

The rewriting of q w.r.t. $\mathcal{T}$ consists of q and the following queries :

$$q' = \{\mathsf{WorkOfArt}(?x)\} \tag{8.4}$$

$$q'' = \{\mathsf{Painting}(?x)\} \tag{8.5}$$

Algorithm 7 summarizes the strategy followed for the implementation of query answering over cultural data. The input to the system is a conjunctive instance query q, given by the user in SPARQL, and the queried ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, i.e., the semantic repository and the relevant knowledge from the EDM ontology, the domain metadata standards and the thematic ontologies. The output of the system is the set of answers of q over $\mathcal{O}$, i.e., all the mappings of query variables to individuals of the semantic repository (the individuals of the ABox $\mathcal{A}$) that satisfy the restrictions of the query and the ontology $\mathcal{O}$.

Let us now describe the functionality of the system. In the beginning, the call of the procedure findOWLQLTerm($\mathcal{T}$) results in the computation of $\mathcal{T}_{QL}$ that is the maximal subset of the terminology $\mathcal{T}$ containing only DL-Lite$_R$ axioms. Then, with the aid of a rewriting algorithm rewrQA, all the rewritings Q_r of q in terms of $\mathcal{T}_{QL}$ are computed, then executed over the ABox $\mathcal{A}$, with the aid of execute and the set Ans of correct answers is computed and given to the user. Obviously, Ans is not the complete query answer set if $\mathcal{T} \setminus \mathcal{T}_{QL} \neq \emptyset$, so in this case, we call the query answering engine evaluate that finally computes all the correct answers based on Algorithm 4 of Chapter 6. The strong point of this hybrid query answering system is its ability to work progressively, i.e., to compute first query answers that come from the QL fragment of OWL and are easier to compute and then answers from the OWL DL fragment that are more difficult to be computed. Note that the answers returned by the rewriting method are not re-checked by method evaluate.

8.2 Experimental Evaluation of the System

The implemented system first uses the metadata aggregation subsystem to aggregate the content provided to Europeana by different providers. We focus on the hellenic

Algorithm 7 evaluateCulturalQueries

Input: $\mathcal{O}$: the queried ontology $\langle \mathcal{T}, \mathcal{A} \rangle$
q: a conjunctive instance query q
Output: Ans: the answers of q over $\mathcal{O}$

```
1: T_QL = findOWLQLTerm(T)
2: Q_r := {rewrQA(q, T_QL)}
3: Ans := {execute(Q_r, A)}
4: if T \ T_QL ≠ ∅ then
5:     Ans := Ans ∪ {evaluate(O, q)}
6: end if
```

content in Europeana, provided through the Athena project,[10] since it is for this content that we possess thematic knowledge. This knowledge is used to illustrate the performance of the proposed semantic query answering methodology.

This content has been transformed to LIDO XML format. Each of the LIDO records represents a museum object (proxy instance) and is described among others by an identifier, a type, a description, the material it is made of, the museum where it can be found, the date it was created. All this information is given as data values (strings) of LIDO elements. In particular, this cultural content is classified in 55 categories (such as pottery, jewelry, stamps, wall paintings, engravings, coins) and more than 300 types, within 17 time periods from 35,000 b.c. up to today.

In the following, 40,000 of the—provided to Europeana—hellenic objects have been included in our study, with an equivalent amount of more than one million (1,000,000) RDF triples being generated and used for query answering. Using the metadata aggregation subsystem, the LIDO XML records were transformed to EDM RDF (were mapped to the EDM ontology). However, this mapping does not suffice for reasoning over these data, because the EDM ontology contains only general axioms about the concepts and roles that describe the records. Moreover, data values - strings are used for the description of objects, which do not allow for complex reasoning. For example, the field that refers to the type of a cultural object is given as a string and, hence, cannot be efficiently exploited by reasoning procedures.

To achieve semantic access, thus providing representations that can be exploited by reasoners, we used the thematic knowledge for hellenic monuments created in the framework of the Polemon project of the Directorate of the National Archive of Monuments,[11] which has been included in the Polydefkis terminology thesaurus of Archaeological Collections and Monuments [12, 23, 69]. Polydefkis is a terminology thesaurus that adopts a classification of objects according to their usage, operation, material they are made of, appearance and decoration. Based mainly on usage, a large number of objects and monument types has been classified accordingly.

In the following, we focus on the part of this knowledge referring to types of vases, since metadata and photos of vases were provided by most above-mentioned hellenic content providers to Europeana through the metadata aggregation subsystem. In particular, the knowledge used contains axioms about vases in ancient Greece, i.e., concept inclusion axioms referring to the different types of vases, such as amphora, alabaster, crater, as well as axioms regarding the appearance, usage, creation period

[10] Access to cultural heritage networks across Europe
[11] `http://nam.culture.gr`

and the material vases were made of. An excerpt from this knowledge (in Description Logic syntax) mainly focusing on the use of vases is provided in Table 8.1.

Table 8.1: *Excerpt of the used thematic ontology in Description Logic syntax*

$\mathsf{Amphora} \sqsubseteq \mathsf{BigVase} \sqcap \mathsf{ClosedVase}$
$\mathsf{Alabaster} \sqsubseteq \mathsf{VaseWithoutHandles}$
$\mathsf{Crater} \sqsubseteq \exists \mathsf{hasBase.NarrowBase}$
$\mathsf{Pycnometer} \sqsubseteq \exists \mathsf{hasBody.CylindricalBody}$
$\mathsf{Amphora} \neq \mathsf{Alabaster}$
$\mathsf{Bowl} \sqsubseteq \mathsf{OpenVase}$
$\mathsf{Lecythus} \sqsubseteq \mathsf{VaseWithoutHandles} \sqcup \mathsf{VaseWithOneHandle}$
$\mathsf{EnclosedProduct} \sqsubseteq \mathsf{Solid} \sqcup \mathsf{Liquid}$
$\mathsf{Solid} \neq \mathsf{Liquid}$
$\mathsf{DrinkingLiquid} \sqsubseteq \mathsf{Liquid}$
$\mathsf{Water} \sqsubseteq \mathsf{DrinkingLiquid}$
$\mathsf{Perfume} \sqsubseteq \mathsf{Liquid}$
$\mathsf{Cereal} \sqsubseteq \mathsf{Solid}$
$\mathsf{Usage} \equiv \mathsf{Carrying} \sqcup \mathsf{Storing} \sqcup \mathsf{Drinking}$
$\exists \mathsf{contains}^{-}.\top \sqsubseteq \mathsf{EnclosedProduct}$
$\exists \mathsf{isUsedFor}^{-}.\top \sqsubseteq \mathsf{Usage}$
$\mathsf{Alabaster} \sqsubseteq \exists \mathsf{isUsedFor.Carrying} \sqcap \exists \mathsf{contains}(\mathsf{Oil} \sqcup \mathsf{Perfume})$
$\mathsf{Amphora} \sqsubseteq \exists \mathsf{isUsedFor.Carrying} \sqcup \exists \mathsf{isUsedFor.Storing}$
$\mathsf{Aryballos} \sqsubseteq \exists \mathsf{contains.Perfume}$
$\mathsf{Cup} \sqsubseteq \exists \mathsf{isUsedFor.Drinking}$
$\mathsf{Hydria} \sqsubseteq \exists \mathsf{isUsedFor.Carrying} \sqcap \exists \mathsf{contains.Water}$
$\mathsf{Vase} \sqcap \exists \mathsf{isUsedFor.Storing} \sqsubseteq \mathsf{StorageVase}$
$\mathsf{Vase} \sqcap \exists \mathsf{madeIn.ArchaicPeriod} \sqsubseteq \mathsf{ArchaicVase}$
$\exists \mathsf{isUsedFor.Storing} \sqcap \exists \mathsf{contains.Liquid} \sqsubseteq \mathsf{LiquidStorageVase}$

After the creation of the above described thematic ontology, the EDM instances were mapped to terms of this ontology. In particular, from the data values appearing in the range of some roles, individual IRIs were created and after being connected (through roles) to proxy instances they were added to the ontology. These were further linked to concepts and roles of the ontology. The creation of individual IRIs and their mapping to the thematic ontology was done using string matching and stemming on the fields of the EDM ontology regarding the type, creation date, material and museum in which proxy instances are found. The OWL API [55] has been used for the creation of the thematic ontology and for the parsing and processing of the EDM RDF data. For some data values, proxy instances were directly assigned to concepts of the ontology. For example, each proxy has been put as an instance of one vase type. As far as the creation date of objects is concerned, time was split to periods of particular interest and each proxy instance was assigned to one of these periods according to the value in the appropriate field of the EDM RDF data. The resulting tuples of this procedure were then added in a Sesame[12] repository.

Using the above described ontologies and data sets, we applied the methodology described in Section 8.1 to provide answers to queries. In particular, the procedure

[12] `http://www.openrdf.org/`

Table 8.2: *Response times (ms) of the two query answering methods and system results*

Query	Time(1)	Time(2)	Results	Results \ Reas
1. $q = \{\mathsf{Bowl}(?x)\}$	145	3,765	127	127
2. $q = \{\mathsf{OpenVase}(?x)\}$	165	13,080	404	0
3. $q = \{\mathsf{Vase}(?x), \mathsf{madeBy}(?x, ?y), \mathsf{Clay}(?y), \mathsf{madeIn}(?x, ?z), \mathsf{CopperPeriod}(?z)\}$	295	15,911	348	0
4. $q = \{\mathsf{VaseWithoutHandles}(?x)\}$	217	16,859	357	0
5. $q = \{\mathsf{ArchaicAmphora}(?x), \mathsf{isUsedFor}(?x, ?y), \mathsf{Storing}(?y)\}$	223	27,945	19	0

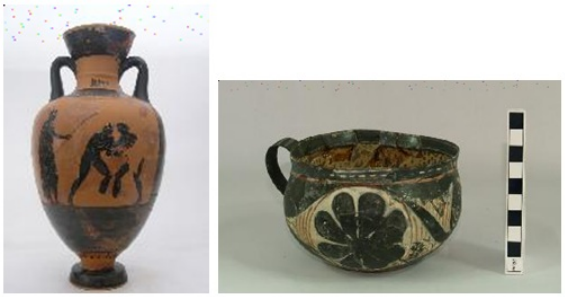

Figure 8.3: *A closed vase (on the left) and an open vase (on the right)*

rewrQA is performed by the Rapid system [21], a goal-oriented rewriting system. The procedure evaluate is performed by the implemented system described in Algorithm 4 of Chapter 6. All experiments were performed on a Windows 7 machine with a double core 2.53GHz Intel x86 64 bit processor and Java 1.6.

A sample of the tested queries together with their running times are shown in Table 8.2. The second column refers to the running times of the rewrQA and execute methods of Section 8.1, while the third column refers to the running time of the method evaluate of Section 8.1. Table 8.2 does not show the total running time of our system, since it progressively provides the results as they are computed by the two methods. The fourth column shows the number of results of our system, while the fifth column shows the number of results of the system when we do not use reasoning.

We start with nearly database/triple store queries that do not need any reasoning to get answered but involve only a retrieval task from the repository and continue

Figure 8.4: *A vase without handles (left), with one handle (middle) and with two handles (right)*

with queries that make use of knowledge that is expressible in OWL 2 DL ($\mathcal{SROIQ}$). In particular, Query 1 is matched to triples that are explicitly found in the triple store without any reasoning taking place. Queries 2,3,4,5 all require the use of reasoning. For Queries 2,3 we can take all the answers from the query rewriting technique, i.e., the query rewriting approach is complete for these queries. Queries 4 and 5 use some OWL 2 DL axioms of the created thematic ontology, which are not expressible in DL-Lite$_R$. In this case, we use the query answering approach described in Algorithm 4 of Section 6.1 to retrieve complete answers. The query rewriting technique does not return all answers for these queries, i.e., it is incomplete. Looking at the time it takes to answer the queries, it is evident that the query rewriting technique scales much better for larger amounts of data. However, as we stated before it is incomplete for some queries. The query ordering optimizations presented in Chapter 4 have an effect on the answering times of Queries 3 and 5, which contain more than one atoms. In particular, static ordering produces good results in both cases. It is important to notice that, without the use of the thematic ontology and the proposed semantic query answering system, much fewer results would be obtained, as shown in Table 8.2 (Results \ Reas). Figure 8.3 shows an example of a closed and an open vase; the latter is included in the results of Query 2 of Table 8.2, while Figure 8.4 shows examples of vases with zero, one and two handles; the first is included in the results of Query 4 of Table 8.2. All examples shown can be found in the website of the Hellenic Ministry of Culture and Tourism.[13]

As future work one could explore whether techniques that are currently used for query answering over expressive knowledge using OWL RL reasoning systems can be used with OWL QL systems. For example, one could explore how the use of a query rewriting algorithm over a transformation of the queried OWL DL ontology could lead to the production of a small over-approximation set of query answers over the OWL DL ontology as is done in OWL RL [143, 144]. In this way, only the mappings in the over- minus the under-approximation set remain to be checked by method evaluate. Moreover, work that has been done in ontology repair for repairing an OWL DL ontology $\mathcal{O}$ for a given incomplete OWL RL reasoner over $\mathcal{O}$ may be relevant [132]. For example, one could explore if and under which conditions repairing an OWL DL ontology $\mathcal{O}$ could make an (incomplete) OWL QL reasoner over $\mathcal{O}$, complete over $\mathcal{O}$ plus some additional axioms.

Before closing this chapter, it is worth noting that the images of the cultural objects are usually provided by content providers apart from their metadata. Since, the knowledge given by content providers (in the form of metadata) may be incomplete, for example, there may be a vase for which it is not specified or it cannot be concluded (by the use of the metadata and ontology) whether it has zero, one or two handles, the images of cultural objects can be exploited to extract additional information. In this direction, work has been performed that combines the described query answering module with machine learning techniques, in particular, Support Vector Machines (SVMs) [52], to, more accurately, answer user queries using features extracted from the images of the cultural objects [79]. In particular, we have used the MPEG-7 features [19] together with the SURF features (Speeded-Up Robust Features) [8] of cultural images extracted by the use of image processing algorithms to train SVMs so that they learn to classify the objects to specific categories like vase types (amphora, bowl, hydria) or vase characteristics (open vase,

[13] http://collections.culture.gr/

vase with two handles); we have trained one SVM for each concept of the queried ontology. The input to each SVM is a two-dimensional matrix, each value of which represents the similarity between a pair of images from the training set based on their visual feature vector; as a similarity measure we have used a normalized linear kernel function [117, 14]. The output of each SVM is a category/concept, to which the cultural object should be classified. Afterwards, the trained SVMs are used to classify new cultural objects; each SVM can decide whether an object belongs or does not belong to a category based on its visual features which are given as input to the SVM. The answer to a concept instance retrieval query is computed by iterating over all objects and taking those objects that the respective SVM classified to the query concept. The answer to a concept query (containing a conjunction of concept atoms) according to the SVM is then taken after joining the query answers for each query concept atom. In the end, the answer to a concept query is computed by taking the union of the answers provided by the SVM and the answers provided by Algorithm 4 of Section 6.1 using the ontology and metadata that are described above. For more information we refer interested readers to [79].

Chapter 9

Conclusions and Future Work

Query answering in the context of ontologies has gained much attention during the last couple of years. Because of the high worst case complexity of answering conjunctive instance queries over ontologies represented in Description Logics, current methods targeting the development of practical systems mainly follow two distinct directions. The first direction suggests reduction of the expressivity of the ontology language used for the representation of the domain knowledge, while the second sacrifices completeness of the query answering process, providing as much expressivity as it is needed. Up to now, there is not much work in optimizing query answering over ontologies in expressive Description Logics, like $\mathcal{SROIQ}$.

9.1 Conclusions and Significance of the Work

In this thesis, we aimed at bridging the gap between high expressivity of the queried ontological knowledge and practicality of query answering over this knowledge. The main goal was the development of a query answering algorithm together with optimizations for SPARQL-OWL query answering over ontologies in expressive languages. SPARQL-OWL is a very expressive query language, which allows queries with variables in place of concept and role names apart from individuals. To the best of our knowledge, there is no work that aims at optimizing answering such complex queries.

In the beginning, we have introduced related work and analyzed existing techniques and optimizations for answering conjunctive instance queries. Afterwards, we have focused on devising and optimizing an algorithm for efficiently evaluating queries. In order to achieve this, the following tasks were performed:

- *Development of a query evaluation algorithm for the OWL 2 Direct Semantics entailment regime of SPARQL*: The SPARQL OWL entailment regime only specifies the set of valid mappings for query variables giving implementors the freedom to decide how they will compute the answers using a great variety of different procedures such as tableau, hypertableau, resolution. In the current thesis we have chosen to use hypertableau calculi and have defined an efficient way to evaluate queries by taking into account relations that exist between axiom templates. The presented algorithm is, nevertheless, more generally applicable and, instead of hypertableau, other reasoning calculi can be used.

- *Definition of a cost model for ordering the axiom templates of queries*: We

have developed a procedure to search for efficient query execution plans. The possible plans for a query are distinguished in terms of their costs, which are determined by means of a cost function and, based on this, the plans are placed in an order of increasing cost. The computation of the costs is based on information about the known and possible instances of concepts and roles and the known and possible equivalent individuals. This information is extracted from a model abstraction built by an OWL reasoner. We have presented two cost functions, a static and a dynamic one and, based on these functions, we have devised algorithms for finding optimal or near optimal query execution orders. According to static ordering, the costs (and the execution order) are determined in the beginning before the actual query evaluation takes place, whereas, in the dynamic case, the costs are computed in parallel with query evaluation, taking into account the results of the execution of previous axiom templates in the plan. For the dynamic case we have improved the performance by exploiting an individual clustering approach that allows for computing the cost functions based on one individual sample per cluster. Moreover, we have explained that the algorithm for the extraction of the known and possible sets of concept and role instances can be seen as an approximate instance retrieval algorithm, in which, the known instances are an under-approximation and the possible instances are an over-approximation of the answers to a query atom. This means that any approximate instance retrieval algorithm can be used in place of the instance retrieval algorithm used in the thesis for answering queries with the proposed query answering and optimization techniques. Moreover, we have shown how our instance retrieval algorithm can be used to define a query answering algorithm.

- *Development of optimizations for queries with complex axiom templates*: We have developed optimizations that exploit the concept and the role hierarchies of the queried ontologies in order to reduce the number of consistency checks performed during the evaluation of complex axiom templates. For this reason, we have defined the notion of polarity of concept and role variable occurrences within axiom templates and proved that, depending on the polarity, we can traverse the concept and role hierarchies in such a way, that some possible solution mappings can be pruned, reducing in this way the number of expensive consistency checks that need to be performed.

- *Development of a system for efficiently answering SPARQL-OWL queries and evaluation of this system*: The proposed approach has been implemented and a system has been developed that can be used for efficiently answering SPARQL-OWL queries over OWL 2 DL ontologies.[1] The system has been implemented as a SPARQL wrapper that can be used with any reasoner that implements the OWLReasoner interface of the OWL API [55]. The developed system has already started to be used; the TrOWL reasoning framework uses the SPARQL wrapper to provide SPARQL support. Using this system, we have evaluated the efficiency of the proposed query ordering and optimization techniques. From our experimental study it has become obvious that the static ordering usually outperforms the dynamic one when accurate statistics are available.

[1] `https://code.google.com/p/owl-bgp/`

This changes, however, when the statistics are less accurate, e.g., due to non-deterministic reasoning decisions. The performance of the system is usually improved when cluster-based sampling is used during query evaluation. For queries that go beyond conjunctive instance queries, we observe an improvement of up to three orders of magnitude when the proposed optimizations are used.

- *Use of the developed system in a practical application*: The developed system has been used for answering queries over a large amount of data coming from the cultural heritage domain.

9.2 Future Work

The work in the current thesis can be extended in several directions. In every case the goal is to further reduce the number of consistency checks performed during the query evaluation procedure.

First of all, the approximate query answering algorithm that has been described in Section 4.5 to quickly find sound answers to conjunctive instance queries can easily be extended to a sound and complete query answering algorithm, as shown in Algorithm 8 (evaluateIntersecQans). Algorithm 8 takes as input an ontology $\mathcal{O}$ and a query q and, using Algorithm intersecQans from Section 4.5, it returns a set of solutions that consists of the known solutions for q, i.e., the set $K_t[q]$ and those possible solutions that lead to the entailment of the instantiated query by $\mathcal{O}$. It is interesting to see how efficient this algorithm is when applied to real life ontologies. In general, we expect that this algorithm will have similar performance results with the query answering algorithm presented in the thesis for conjunctive instance queries that contain atoms with only known instances, since, in this case, both algorithms find answers to such queries by joining the known instances of every atom in each query. For the case of queries containing atoms with possible instances the picture is less clear. On the one hand, the sequential joining and creation of answers for sub-execution plans of the chosen query execution plan, as presented in the thesis, results in smaller intermediate joins; we determine which possible instances are real ones by performing consistency checks. On the other hand, the reduction in the number of possible instances of query atom concepts or roles that need to be checked, due to the fact that some of these possible instances do not join with (known or possible) instances of next query atoms in the execution plan, may lead to better execution times for some queries and ontologies. Let us consider Example 10 from Section 4.5. On the one hand, using Algorithm 8, we avoid performing a consistency check for the mapping $\{?x \mapsto b, ?y \mapsto e\}$, which we would have checked if Algorithm 4 had been used and b was an instance of A. On the other hand, if b was not an instance of A, then the evaluation of the first atom, $A(?x)$, would prune this (possible) mapping and, hence, this mapping would not further be joined with the mappings for the next query atoms in the execution plan, something which would have been done by Algorithm 8. Hence, it would be interesting to study this trade-off between the increase in the number of joins needed to evaluate a query that comes with a possible decrease in the number of possible instances that need to be checked and the decrease in the number of performed joins that may come with an increase in the number of possible instances that need to be

Algorithm 8 evaluateIntersecQans($\mathcal{O}, q$)

Input: $\mathcal{O}$: the queried $\mathcal{SROIQ}$ ontology
q: a conjunctive instance query over $\mathcal{O}$
Output: a set of solutions for evaluating q over $\mathcal{O}$
1: $\langle K_t[q], P_t[q] \rangle :=$ intersecQans($\mathcal{O}, q$)
2: $R_{ans} := \{\mu \mid \mu \in K_t[q]\} \cup \{\mu \mid \mu \in P_t[q], \mathcal{O} \models \mu(q)\}$
3: **return** R_{ans}

checked.

In this direction another technique can be explored that may further reduce the possible instances in the evaluation of conjunctive instance queries. Although this technique increases the number of performed joins between query atoms as it adds additional atoms to the query, we generally expect that it will lead to better execution times as it can significantly reduce the number of possible instances of atoms. According to this technique, the initial query is extended, using the TBox, with additional atoms that do not affect the answers to this query over any dataset, and which may contain useful information for reducing the possible instances of query concepts and roles. Note that this reduction is query specific, i.e., the reduced sets of known and possible instances of query concepts and roles are valid only for the given query. This optimization is useful not only for the actual query evaluation but also for the creation of better estimates for query ordering. This is possible since, using this technique, the results of joins of query atoms can, more accurately, be taken into account. In more detail, assuming that an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and a conjunctive instance query q is given, the steps that can be followed according to this technique are the following:

1. The variables in the initial query q are replaced by fresh individual names producing a corresponding query ABox.

2. Materialization is afterwards performed over the queried TBox and the (small) query ABox constructed in Step 1 and the entailed concept and role name assertions constitute the extended query ABox.

3. The individual names from the extended query ABox (that were used for variable substitutions in Step 1) are then replaced with the corresponding variables from q and a new query $\hat{q}$ is produced that is equivalent with q. This means that, for any ABox $\mathcal{A}$, the answers of q over $\langle \mathcal{T}, \mathcal{A} \rangle$ are the same as the answers of $\hat{q}$ over $\langle \mathcal{T}, \mathcal{A} \rangle$.

4. The additional query atoms in $\hat{q}$ can then be used for reducing the sets of possible instances to which query variables are mapped.

The performance of this and the previous approach should be experimentally tested and compared or possibly interweaved with the approach developed in the current thesis. A preliminary work on the above optimizations has already been performed [35].

The absence of suitable benchmarks restricts the thorough evaluation and testing of the techniques developed for query answering. In the thesis, the limited number of existing benchmarks [45, 88] have been used for testing the query answering techniques for conjunctive instance queries. Moreover, queries developed

for SPARQL-DL have been used over existing ontologies to test the efficiency of the presented techniques. In order to test the effect of the proposed optimizations for the case of queries with complex axiom templates, we have manually created queries for two ontologies.

In order to give scientists working on query answering over ontologies the ability to test the techniques they propose with a greater number and more interesting benchmarks, an important future work is the generation of near-realistic ontologies written in expressive Description Logics that can be used for testing a system's scalability. Several TBoxes in expressive DLs have been developed in the last couple of years. However, these usually have very small ABoxes or none at all and no queries are usually given for them. Hence, we are currently looking at techniques that, given a TBox in an expressive Description Logic and (possibly) a small representative ABox, they try to increase the size of the ABox in such a way that this ABox expansion is done in an interesting way. We say "in an interesting way", because, for example, a simple multiplication of a given ABox would not be sufficient as tools for query answering are likely to resort to partitioning techniques. In order to avoid this, transformations need to be developed that modify the replicated ABoxes in such a way that certain properties of the original ABox are preserved. For example, it is reasonable to preserve the consistency of the original ABox or it may be interesting to preserve non-empty answers to queries.

Apart from the above described more concrete future work it would also be interesting to integrate the techniques presented in the thesis with more techniques and optimizations from the area of databases, such as the use of magic sets [11] or the use of indexes for the efficient retrieval of known and possible instances or answers. Moreover, since the presented approach can be used to develop an approximate query answering algorithm, it would be interesting to perform experiments and compare the performance of this approach and other approximate query answering systems proposed in the literature.

Bibliography

[1] OBO and OWL: Leveraging semantic web technologies for the life sciences. In *Proceedings of the 6th International and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, 2007.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1994.

[3] Andrea Acciarri, Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati. Quonto: Querying ontologies. In *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pages 1670–1671, 2005.

[4] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-lite family and relations. *J. Artif. Int. Res.*, 36(1):1–69, September 2009.

[5] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 364–369, 2005.

[6] Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality restrictions on concepts. *Artif. Intell.*, 88(1-2):195–213, 1996.

[7] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, second edition, 2007.

[8] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, June 2008.

[9] Sean Bechhofer and Carole Goble. Description logics and multimedia - applying lessons learnt from the galen project. In *Proceedings of the workshop on Knowledge Representation for Interactive Multimedia Systems (KRIMS'96), at ECAI'96*, 1996.

[10] Dave Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers, editors. *Turtle – Terse RDF Triple Language*. W3C Working Draft, 10 July 2012. Available at `http://www.w3.org/TR/turtle/`.

[11] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. In *Proceedings of the Sixth ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 269–284. ACM, 1987.

[12] Ch. Bekiari, Ch. Gritzapi, and D Kalomirakis. POLEMON : A Federated Database Management System for the Documentation, Management and Promotion of Cultural Heritage. In *Proceedings of the 26th Conference on Computer Applications in Archaeology*, 1998.

[13] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[14] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[15] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[16] Dan Brickley and Ramanathan V. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-schema/`.

[17] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[18] Diego Calvanese, Guiseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

[19] S. F. Chang, T. Sikora, and A. Puri. Overview of the MPEG-7 standard. *IEEE Trans. Circuits and Systems for Video Technology*, 11(6):688–695, June 2001.

[20] Surajit Chaudhuri. An Overview of Query Optimization in Relational Systems. In *Proceedings of the Seventeenth ACM SIGMOD Symposium on Principles of Database Systems*, pages 34–43. ACM Press, 1998.

[21] Alexandros Chortaras, Despoina Trivela, and Giorgos Stamou. Optimized query rewriting for OWL 2 QL. In *Proceedings of the 23rd International Conference on Automated Deduction*, CADE'11, pages 192–206. Springer-Verlag, 2011.

[22] Mike Dean and Guus Schreiber, editors. *OWL Web Ontology Language Reference*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-ref/`.

[23] D.Kalomirakis and A.Alexandri. Deploying the POLEMON system for the National Monuments Record of Greece: experience and outlook. In *Computer Applications and Quantitative Methods, Archaeology Conference*. CEUR Electronic Workshop Proceedings, 2002.

[24] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Edith Schonberg, Kavitha Srinivas, and Li Ma. Scalable semantic retrieval through summarization and refinement. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, AAAI*, pages 299–304, 2007.

[25] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Li Ma, Edith Schonberg, Kavitha Srinivas, and Xingzhi Sun. Scalable Grounded Conjunctive Query Evaluation over Large and Expressive Knowledge Bases. In *Proceedings of the 7th International Semantic Web Conference*, pages 403–418, 2008.

[26] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Edith Schonberg, and Kavitha Srinivas. Scalable highly expressive reasoner (SHER). *Journal of Web Semantics*, 7(4):357–361, 2009.

[27] Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query Rewriting for Horn-SHIQ Plus Rules. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012*.

[28] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 2nd edition, 1996.

[29] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.

[30] Birte Glimm. Querying Description Logic Knowledge Bases. Doctoral thesis, University of Manchester, 2007.

[31] Birte Glimm. Using SPARQL with RDFS and OWL Entailment. In *Reasoning Web, Semantic Technologies for the Web of Data - 7th International Summer School*, pages 137–201, 2011.

[32] Birte Glimm, Ian Horrocks, Carsten Lutz, and Uli Sattler. Conjunctive query answering for the description logic SHIQ. *Journal of Artificial Intelligence Research*, 31:151–198, 2008.

[33] Birte Glimm, Ian Horrocks, Boris Motik, Rob Shearer, and Giorgos Stoilos. A novel approach to ontology classification. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 14:84–101, 2012.

[34] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Unions of conjunctive queries in $\mathcal{SHOQ}$. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 252–262, 2008.

[35] Birte Glimm, Yevgeny Kazakov, Ilianna Kollia, and Giorgos B. Stamou. Using the TBox to Optimise SPARQL Queries. In *Proceedings of the 25th International Workshop on Description Logics (DL-2013)*, CEUR Workshop Proceedings, 2013.

[36] Birte Glimm and Markus Krötzsch. SPARQL beyond subgraph matching. In *Proceedings of the 9th International Semantic Web Conference (ISWC'10)*. Springer, 2010.

[37] Birte Glimm and Chimezie Ogbuji. SPARQL 1.1 entailment regimes. W3C Recommendation, 21 March 2013. Available at `http://www.w3.org/TR/sparql11-entailment/`.

[38] Christine Golbreich, Songmao Zhang, and Olivier Bodenreider. The foundational model of anatomy in OWL: Experience and perspectives. *Journal of Web Semantics*, 4(3):181–195, 2006.

[39] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Ontological Queries: Rewriting and Optimization. In *Proceedings of the 27th International Conference on Data Engineering*, ICDE '11, pages 2–13, 2011.

[40] Georg Gottlob and Thomas Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning, KR 2012*.

[41] Anastasios Gounaris, Norman W. Paton, Alvaro A.A. Fernandes, and Rizos Sakellariou. Adaptive query processing: A survey. In *Advances in Databases, 19th British National Conference on Databases, BNCOD*, pages 11–25. Springer, 2002.

[42] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Journal of Web Semantics*, 6(4):309–322, 2008.

[43] Bernardo Cuenca Grau, Boris Motik, Giorgos Stoilos, and Ian Horrocks. Completeness guarantees for incomplete ontology reasoners: Theory and practice. *Journal of Artificial Intelligence Research (JAIR)*, 43:419–476, 2012.

[44] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *Proceedings of the 12th International Conference on World Wide Web (WWW'03)*, pages 48–57. ACM, 2003.

[45] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.

[46] V. Haarslev, R. Möller, and A.Y. Turhan. Exploiting Pseudo Models for TBox and ABox Reasoning in Expressive Description Logics. In *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR'01)*, pages 29–44, 2001.

[47] Volker Haarslev and Ralf Möller. Racer system description. In *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR'01)*, pages 701–705, 2001.

[48] Volker Haarslev and Ralf Möller. On the scalability of description logic instance retrieval. *Journal of Automated Reasoning*, 41(2):99–142, 2008.

[49] Volker Haarslev, Ralf Möller, and Michael Wessel. Querying the semantic web with Racer + nRQL. In *Proceedings of the KI-2004 International Workshop on Applications of Description Logics*, 2004.

[50] Steve Harris and Andy Seaborne, editors. *SPARQL 1.1 Query Language*. W3C Proposed Recommendation, 08 November 2012. Available at `http://www.w3.org/TR/sparql11-query/`.

[51] Patrick Hayes, editor. *RDF Semantics*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-mt/`.

[52] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[53] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2009.

[54] Pascal Hitzler and Denny Vrandecic. Resolution-based approximate reasoning for OWL DL. In *Proceedings of the 4th International Semantic Web Conference (ISWC 2005)*, pages 383–397, 2005.

[55] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for working with OWL 2 ontologies. In *Proceedings of the OWLED 2009 Workshop on OWL: Experiences and Directions*, 2009.

[56] Matthew Horridge and Peter F. Patel-Schneider, editors. *OWL 2 Web Ontology Language: Manchester Syntax*. W3C Working Group Note, 27 October 2009. Available at `http://www.w3.org/TR/owl2-manchester-syntax/`.

[57] Ian Horrocks. Optimising tableaux decision procedures for description logics. Doctoral thesis, University of Manchester, 1997.

[58] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 57–67.

[59] Ian Horrocks and Ulrike Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.

[60] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for SHOIQ. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 448–453, 2005.

[61] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning*, LPAR '99, pages 161–180, 1999.

[62] Ian Horrocks and Stephan Tobies. Reasoning with axioms: Theory and practice. In *Proc. of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 285–296, 2000.

[63] Hai Huang and Chengfei Liu. Estimating selectivity for joined rdf triple patterns. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11*, pages 1435–1444, 2011.

[64] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.

[65] Ulrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing $\mathcal{SHIQ}^-$ description logic to disjunctive datalog programs. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, pages 152–162. AAAI Press, 2004.

[66] Yannis Ioannidis. The history of histograms (abridged). In *Proc. of VLDB Conference*, 2003.

[67] Yannis E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 28(1):121–123, March 1996.

[68] Matthias Jarke and Jürgen Koch. Query optimization in database systems. *ACM Computing Surveys*, 16:111–152, 1984.

[69] D. Kalomirakis and A. Kalatzopoulou. Polydefkis: A Terminology Thesauri for Monuments. In *Applications of Advanced Technology in Archaelogical Research and Spilling of its Results*, 2000.

[70] Zoi Kaoudi, Kostis Kyzirakos, and Manolis Koubarakis. SPARQL Query Optimization on Top of DHTs. In *Proceedings of the 9th International Semantic Web Conference*, pages 418–435, 2010.

[71] Alissa Kaplunova, Ralf Möller, Sebastian Wandelt, and Michael Wessel. Towards Scalable Instance Retrieval over Ontologies. In *Proceedings of the 4th International Conference on Knowledge Science, Engineering and Management, KSEM 2010*, pages 436–448, 2010.

[72] Yevgeny Kazakov. $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 274–284. AAAI Press, 2008.

[73] Ilianna Kollia and Birte Glimm. Cost Based Query Ordering over OWL Ontologies. In *Proceedings of the 24th International Workshop on Description Logics (DL-2012)*, CEUR Workshop Proceedings, 2012.

[74] Ilianna Kollia and Birte Glimm. Cost Based Query Ordering over OWL Ontologies. In *Proceedings of the 11th International Semantic Web Conference*, volume 7649 of *Lecture Notes in Computer Science*, pages 231–246. Springer, 2012.

[75] Ilianna Kollia and Birte Glimm. Optimizing SPARQL Query Answering over OWL Ontologies. *J. Artif. Intell. Res. (JAIR)*, 48:253–303, 2013.

[76] Ilianna Kollia, Birte Glimm, and Ian Horrocks. Answering Queries over OWL Ontologies with SPARQL. In *Proceedings of the 8th International Workshop on OWL: Experiences and Directions (OWLED)*, volume 796 of *CEUR Workshop Proceedings*, 2011.

[77] Ilianna Kollia, Birte Glimm, and Ian Horrocks. Query Answering over SROIQ knowledge bases with SPARQL. In *Proceedings of the 24th International Workshop on Description Logics (DL 2011)*, CEUR Workshop Proceedings, 2011.

[78] Ilianna Kollia, Birte Glimm, and Ian Horrocks. SPARQL Query Answering over OWL Ontologies. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC'11)*, Lecture Notes in Computer Science, pages 382–396. Springer, 2011.

[79] Ilianna Kollia, Yannis Kalantidis, Kostas Rapantzikos, and Andreas Stafylopatis. Improving Semantic Search in Digital Libraries Using Multimedia Analysis. *Journal of Multimedia*, 7(2):193–204, 2012.

[80] Ilianna Kollia, Kostas Rapantzikos, Giorgos B. Stamou, and Andreas Stafylopatis. Semantic Query Answering in Digital Libraries. In *Proceedings of the 7th Hellenic Conference on Artificial Intelligence (SETN)*, Lecture Notes in Computer Science, pages 17–24. Springer, 2012.

[81] Ilianna Kollia, Vassilis Tzouvaras, Nasos Drosopoulos, and Giorgos B. Stamou. A Systemic Approach for Effective Semantic Access to Cultural Content. *Semantic Web Journal*, 3(1):65–83, 2012.

[82] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to query answering in DL-Lite. In *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR'10)*. AAAI Press, 2010.

[83] Petr Kremen. Building Ontology-Based Information Systems. Doctoral thesis, Czech Technical University in Prague, 2012.

[84] Petr Kremen and Evren Sirin. SPARQL-DL implementation experience. In *Proceedings of the 4th OWLED Workshop on OWL: Experiences and Directions*, volume 496 of *CEUR Workshop Proceedings*, 2008.

[85] Markus Krötzsch. Efficient inferencing for OWL EL. In *Proceedings of the 12th European Conference on Logics in Artificial Intelligence (JELIA'10)*, volume 6341 of *LNAI*, pages 234–246. Springer, 2010.

[86] Markus Krötzsch. OWL 2 Profiles: An Introduction to Lightweight Ontology Languages. In *Reasoning Web, Semantic Technologies for the Web of Data - 8th International Summer School*, volume 7487 of *Lecture Notes in Computer Science*, pages 112–183. Springer, 2012.

[87] Carsten Lutz, David Toman, and Frank Wolter. Conjunctive Query Answering in the Description Logic EL using a relational database system. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2070–2075, 2009.

[88] Li Ma, Yang Yang, Zhaoming Qiu, Guotong Xie, Yue Pan, and Shengping Liu. Towards a complete OWL ontology benchmark. In *The Semantic Web: Research and Applications*, Lecture Notes in Computer Science. Springer, 2006.

[89] Michael V. Mannino, Paicheng Chu, and Thomas Sager. Statistical profile estimation in database systems. *ACM Comput. Surv.*, 20(3):191–221, 1988.

[90] Frank Manola and Eric Miller, editors. *Resource Description Framework (RDF): Primer*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-primer/`.

[91] Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany, January 2006.

[92] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz, editors. *OWL 2 Web Ontology Language: Profiles*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-profiles/`.

[93] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau, editors. *OWL 2 Web Ontology Language: Direct Semantics*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-direct-semantics/`.

[94] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia, editors. *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-syntax/`.

[95] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. In *Proceedings of the 21st Conference on Automated Deduction (CADE'07)*, volume 4603 of *LNAI*, pages 67–83. Springer, 2007.

[96] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.

[97] B. Neumann and R. Möller. On scene interpretation with description logics. Technical Report FBI-B-257/04, Fachbereich Informatik, Universität Hamburg, 2004.

[98] Thomas Neumann and Guido Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *Proceedings of the 27th International Conference on Data Engineering*, ICDE '11, pages 984–994. IEEE Computer Society, 2011.

[99] Thomas Neumann and Gerhard Weikum. The RDF-3X engine for scalable management of RDF data. *The VLDB Journal*, 19(1):91–113, February 2010.

[100] Frank Olken and Doron Rotem. Simple random sampling from relational databases. In *Proceedings of VLDB'86 Twelfth International Conference on Very Large Data Bases*, pages 160–169. Morgan Kaufmann, 1986.

[101] Magdalena Ortiz and Mantas Simkus. Reasoning and query answering in description logics. In *Reasoning Web, Semantic Technologies for Advanced Query Answering - 8th International Summer School*, pages 1–53, 2012.

[102] Jeff Z. Pan and Edward Thomas. Approximating OWL-DL Ontologies. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 1434–1439. AAAI Press, 2007.

[103] Jeff Z. Pan, Edward Thomas, and Yuting Zhao. Completeness Guaranteed Approximation for OWL DL Query Answering. In *Proceedings of the 2009 International Workshop on Description Logics (DL'09)*, 2009.

[104] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[105] Peter F. Patel-Schneider and Boris Motik, editors. *OWL 2 Web Ontology Language: Mapping to RDF Graphs*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-mapping-to-rdf/`.

[106] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):1–45, 2009.

[107] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Efficient Query Answering for OWL 2. In *Proceedings of the 8th International Semantic Web Conference*, pages 489–504, 2009.

[108] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, 8(2):186–209, 2010.

[109] Kosmas Petridis, Stephan Bloehdorn, Carsten Saathoff, Nikos Simou, Stamatia Dasiopoulou, Vassilis Tzouvaras, Siegfried Handschuh, Yannis Avrithis, Yiannis Kompatsiaris, and Steffen Staab. Knowledge representation and semantic annotation of multimedia content. *IEEE Proceedings on Vision, Image and Signal Processing - Special issue on the Integration of Knowledge, Semantics and Digital Media Technology*, 153(3):255–262, 2006.

[110] Eric Prud'hommeaux and Andy Seaborne, editors. *SPARQL Query Language for RDF*. W3C Recommendation, 15 January 2008. Available at `http://www.w3.org/TR/rdf-sparql-query/`.

[111] Alan L. Rector and Ian R. Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proc. of the Workshop on Ontological Engineering*, pages 414–418, 1997.

[112] Mariano Rodriguez-Muro and Diego Calvanese. High performance query answering over dl-lite ontologies. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press, 2012.

[113] Riccardo Rosati. On conjunctive query answering in EL. In *Proceedings of the 2007 International Workshop on Description Logic (DL 2007)*. CEUR Electronic Workshop Proceedings, 2007.

[114] Riccardo Rosati and Alessandro Almatelli. Improving Query Answering over DL-Lite Ontologies. In *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press, 2010.

[115] S. S. Sahoo, O. Bodenreider, K. Zeng, and A. Sheth. An experiment in integrating large biomedical knowledge resources with RDF: Application to associating genotype and phenotype information. In *Workshop on health care and life sciences data integration for the semantic web at the 16th international world wide web conference (WWW, 2007)*. Citeseer, 2007.

[116] Michael Schneider, editor. *OWL 2 Web Ontology Language: RDF-Based Semantics*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-rdf-based-semantics/`.

[117] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[118] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006.

[119] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*.

[120] Eep S. Sidhu, Tharam S. Dillon, Elizabeth Chang, and Baldev S. Sidhu. Protein ontology development using OWL. In *Proceedings of the First OWL Experiences and Directions Workshop*, page 188, 2005.

[121] Abraham Silberschatz, Henry Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, Inc., New York, NY, USA, 5 edition, 2006.

[122] Nicholas Sioutos, Sherri de Coronado, Margaret W. Haber, Frank W. Hartel, Wen-Ling Shaiu, and Lawrence W. Wright. NCI Thesaurus: A Semantic Model Integrating Cancer-related Clinical and Molecular Information. *J. of Biomedical Informatics*, 40(1):30–43, February 2007.

[123] Evren Sirin, Bernardo Cuenca Grau, and Bijan Parsia. From wine to water: Optimizing description logic reasoning for nominals. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 90–99. AAAI Press, 2006.

[124] Evren Sirin and Bijan Parsia. Optimizations for answering conjunctive ABox queries: First results. In *Proceedings of the 2006 International Workshop on Description Logics (DL'06)*, volume 189 of *CEUR Workshop Proceedings*, 2006.

[125] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL query for OWL-DL. In *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, volume 258 of *CEUR Workshop Proceedings*, 2007.

[126] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[127] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J. Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J. Mungall, The OBI Consortium, Neocles Leontis, Philippe Rocca-Serra, Alan Ruttenberg, Susanna-Assunta Sansone, Richard H. Scheuermann, Nigam Shah, Patricia L. Whetzeland, and Suzanna Lewis. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, 25:1251–1255, 2007.

[128] Kent A. Spackman, Ph. D, Keith E. Campbell, Ph. D, Roger A. Côté, and D. Sc. (hon). SNOMED RT: A reference terminology for health care. In *J. of the American Medical Informatics Association*, pages 640–644, 1997.

[129] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and randomized optimization for the join problem. *VLDB Journal*, 6:191–208, 1997.

[130] Robert Stevens, Mikel Egana Aranguren, Katy Wolstencroft, Ulrike Sattler, Nick Drummond, Matthew Horridge, and Alan Rector. Using OWL to model biological knowledge. *International Journal of Human-Computer Studies*, 65(7):583 – 594, 2007.

[131] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. SPARQL basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*, pages 595–604. ACM, 2008.

[132] Giorgos Stoilos, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks. Repairing ontologies for incomplete reasoners. In *Proceedings of the 10th International Semantic Web Conference (ISWC-11)*. Springer, 2011.

[133] Herman J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3(2–3):79–115, 2005.

[134] Edward Thomas, Jeff Z. Pan, and Yuan Ren. TrOWL: Tractable OWL 2 reasoning infrastructure. In *Proceedings of the Extended Semantic Web Conference (ESWC'10)*, 2010.

[135] Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 41–50, 2004.

[136] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *LNCS*, pages 292–297. Springer, 2006.

[137] Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider. Optimizing terminological reasoning for expressive description logics. *Journal of Automated Reasoning*, 39(3):277–316, 2007.

[138] Tuvshintur Tserendorj, Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. Approximate OWL-Reasoning with Screech. In *Proceedings of the Second International Conference on Web Reasoning and Rule Systems, RR 2008*, volume 5341 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2008.

[139] Frank van Harmelen, Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of Knowledge Representation*. Elsevier Science, San Diego, USA, 2007.

[140] Katy Wolstencroft, Andy Brass, Ian Horrocks, Phillip W. Lord, Ulrike Sattler, Daniele Turi, and Robert Stevens. A little semantic web goes a long way in biology. In *Proceedings of the 4th International Semantic Web Conference (ISWC)*, volume 3729 of *Lecture Notes in Computer Science*, pages 786–800. Springer, 2005.

[141] C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, and L. Moreau. Automating experiments using semantic data in a bioinformatics grid. *Intelligent Systems, IEEE*, 19(1):48–55, 2004.

[142] Christopher Wroe, James Cimino, and Alan Rector. Integrating Existing Drug Formulation Terminologies Into an HL7 Standard Classification using OpenGALEN. In *Annual Fall Symposium of American Medical Informatics Association*, Washington DC., November 2001.

[143] Yujiao Zhou, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, and Jay Banerjee. Making the Most of your Triple Store: Query Answering in OWL 2 Using an RL Reasoner. In *Proceedings of the 22nd International World Wide Web Conference (WWW)*, 2013.

[144] Yujiao Zhou, Yavor Nenov, Bernardo Cuenca Grau, and Ian Horrocks. Complete query answering over horn ontologies using a triple store. In *Proc. of the 12th International Semantic Web Conference (ISWC)*. Springer LNCS, 2013.

Περίληψη

Η απάντηση ερωτημάτων που τίθενται σε οντολογίες, δηλαδή η εύρεση απαντήσεων λαμβάνοντας υπόψη όχι μόνο ρητά εκφραζόμενη αλλά και έμμεση πληροφορία που εξάγεται με βάση λογικούς κανόνες είναι μια σημαντική περιοχή έρευνας στο Σημασιολογικό Ιστό. Σε αυτήν την κατεύθυνση, η SPARQL, γλώσσα απάντησης ερωτημάτων, που τυποποιήθηκε το 2008 από τον Οργανισμό Παγκόσμιου Ιστού (World Wide Web Consortium, W3C) επεκτείνεται με τα λεγόμενα συστήματα συνεπαγωγής. Ένα σύστημα συνεπαγωγής προσδιορίζει τον τρόπο που αξιολογούνται τα ερωτήματα χρησιμοποιώντας πιο εκφραστική σημασιολογία από την απλή συνεπαγωγή της SPARQL η οποία στηρίζεται σε ταίριασμα υπογράφων.

Την τελευταία δεκαετία έχει γίνει μεγάλη προσπάθεια για τη βελτιστοποίηση των υπηρεσιών συλλογιστικής, όπως είναι ο έλεγχος συνέπειας μιας οντολογίας και η ταξινόμηση των εννοιών μιας οντολογίας (η εύρεση της ιεραρχίας των εννοιών). Η ανάπτυξη βελτιστοποιήσεων για την απάντηση ερωτημάτων σε οντολογίες έχει μόλις πρόσφατα απασχολήσει την επιστημονική κοινότητα, η οποία έχει ασχοληθεί κυρίως με συζευκτικά ερωτήματα που τίθενται σε υποσύνολα της γλώσσας οντολογίας Ιστού (OWL). Μέχρι στιγμής μεγαλύτερη έμφαση έχει δοθεί στο OWL 2 QL προφίλ.

Στη διατριβή αυτή επικεντρωνόμαστε στο SPARQL σύστημα συνεπαγωγής OWL 2 άμεσης σημασιολογίας (SPARQL-OWL). Το βασικό συστατικό στοιχείο των SPARQL ερωτημάτων είναι τα *βασικά μοτίβα γράφου* (BGPs), δηλαδή τριπλέτες RDF με μεταβλητές, τις οποίες αντιστοιχίζουμε σε γενικευμένα OWL 2 αξιώματα με μεταβλητές σύμφωνα με το σύστημα αυτό. Κάθε τέτοιο αξίωμα αποτελεί ένα άτομο του ερωτήματος. Τα επιτρεπόμενα SPARQL-OWL ερωτήματα είναι πολύ εκφραστικά καθώς πέρα από τις μεταβλητές στη θέση αντικειμένων επιτρέπουν τη χρήση μεταβλητών στη θέση ατομικών εννοιών ή ατομικών ρόλων σε σύνθετες έννοιες και αξιώματα. Μια *απάντηση* σε ένα ερώτημα είναι μια αντιστοίχιση των μεταβλητών που εμφανίζονται στο ερώτημα σε όρους της υπό εξέταση οντολογίας έτσι ώστε η οντολογία να συνεπάγεται (με χρήση της OWL 2 σχέσης συνεπαγωγής άμεσης σημασιολογίας) το ερώτημα (σύνολο OWL αξιωμάτων) που προκύπτει αντικαθιστώντας τις μεταβλητές με τιμές.

Ένας απλοϊκός αλγόριθμος για την απάντηση ερωτημάτων σύμφωνα με το παραπάνω σύστημα συνεπαγωγής ελέγχει, για κάθε έγκυρη αντιστοίχιση όρων της οντολογίας στις μεταβλητές του ερωτήματος, αν η υπό εξέταση οντολογία συνεπάγεται το ερώτημα που προκύπτει. Αν η οντολογία συνεπάγεται το ερώτημα τότε η συγκεκριμένη απόδοση τιμών στις μεταβλητές αποτελεί απάντηση του ερωτήματος αλλιώς όχι.

Επειδή ο σκοπός μας είναι η ανάπτυξη τεχνικών βελτιστοποίησης για την απάντηση ερωτημάτων και οι τεχνικές αυτές δεν ενσωματώνονται εύκολα στον παραπάνω απλοϊκό αλγόριθμο, αρχικά στη διατριβή αυτή περιγράφουμε εναν ορθό και πλήρη αλγόριθμο για το παραπάνω σύστημα συνεπαγωγής ο οποίος εκμεταλλεύεται τις συσχετίσεις που υπάρχουν μεταξύ των ατόμων του ερωτήματος, αξιολογώντας ένα-ένα τα άτομα. Πιο

συγκεκριμένα αρχικά το σύνολο απαντήσεων περιέχει την κενή αντιστοίχιση που δεν αντιστοιχίζει καμιά μεταβλητή σε κάποια τιμή. Στη συνέχεια διαλέγουμε το πρώτο άτομο του ερωτήματος, επεκτείνουμε την κενή αντιστοίχιση ώστε να καλύπτει τις μεταβλητές του επιλεγμένου ατόμου και χρησιμοποιώντας μια μηχανή συλλογιστικής αποφασίζουμε ποιες από τις δυνατές αντιστοιχίσεις οδηγούν σε αξίωμα που συνεπάγεται από την οντολογία. Μετά παίρνουμε το επόμενο άτομο και ομοίως επεκτείνουμε τις αντιστοιχίσεις του προηγούμενου γύρου για να καλύψουμε όλες τις μεταβλητές του νέου ατόμου και ελέγχουμε ποιες από αυτές τις αντιστοιχίσεις οδηγούν σε αξίωμα που συνεπάγεται από την οντολογία. Με τον ίδιο τρόπο συνεχίζουμε την εύρεση απαντήσεων για τα άτομα του ερωτήματος.

Από τα παραπάνω γίνεται σαφές ότι προκειμένου να εκτελεστεί πιο αποδοτικά η παραπάνω διαδικασία είναι απαραίτητη η εύρεση μιας αποδοτικής σειράς εκτέλεσης των ατόμων του ερωτήματος. Οι τεχνικές της εύρεσης βέλτιστων ή σχεδόν βέλτιστων ακολουθιών εκτέλεσης έχουν χρησιμοποιηθεί ευρέως στις βάσεις δεδομένων και στις αποθήκες τριπλετών. Οι τεχνικές αυτές περιλαμβάνουν τη διατήρηση ενός συνόλου από στατιστικές σχετικές με τις σχέσεις που εμφανίζονται σε μια βάση δεδομένων, τους δείκτες που χρησιμοποιούνται, τον αριθμό σελίδων κάποιου δείκτη, τον αριθμό των διαφορετικών τιμών που λαμβάνει μια ιδιότητα μιας σχέσης μαζί με φόρμουλες για τον υπολογισμό της επιλεκτικότητας κατηγορημάτων και του κόστους επεξεργασίας και εισόδου-εξόδου της εκτέλεσης ενός ερωτήματος. Οι τεχνικές αυτές δεν είναι, όμως, άμεσα εφαρμόσιμες στην περίπτωση που έχουμε οντολογίες. Αυτό συμβαίνει επειδή υπό την παρουσία εκφραστικών αξιωμάτων δεν μπορούμε να στηριχτούμε στο πόσες φορές εμφανίζονται στιγμιότυπα σχέσεων στη βάση και επιπλέον δεν μπορούμε να ξέρουμε από την αρχή όλα τα σχετικά συμπεράσματα της οντολογίας από τα οποία θα προκύψουν οι στατιστικές μας. Επίσης ο στόχος μας σε ένα τέτοιο πλαίσιο δεν είναι μόνο η μείωση του πλήθους των αποτελεσμάτων των ατόμων του ερωτήματος αλλά πρέπει επίσης να λαμβάνεται υπόψη το κόστος του ελέγχου ή του υπολογισμού των αποτελεσμάτων. Αυτό το κόστος είναι σημαντικό όταν χρησιμοποιούμε OWL συλλογιστική για την εξαγωγή συμπερασμάτων και επίσης λαμβάνει ένα μεγάλο εύρος τιμών (για παράδειγμα λόγω μη ντετερμινισμού και της υψηλής πολυπλοκότητας χειρότερης περίπτωσης τυπικών διαδικασιών συλλογιστικής). Για αυτό το λόγο η ακριβής εκτίμηση του κόστους αυτού πριν την εκτέλεση του ερωτήματος είναι δύσκολη.

Πιο συγκεκριμένα, η εύρεση ενός πλάνου εκτέλεσης για ένα ερώτημα αφορά στην εύρεση μιας αποτελεσματικής σειράς εκτέλεσης για τα άτομα του ερωτήματος. Τα πλάνα καθορίζονται με τη βοήθεια μιας συνάρτησης κόστους η οποία απονέμει κόστη στα άτομα των ερωτημάτων και στη συνέχεια τα άτομα διατάσσονται σε σειρά αυξανόμενου κόστους. Πιο συγκεκριμένα, η συνάρτηση αυτή εκτιμά δύο συνιστώσες, 1) το κόστος των λειτουργιών συλλογιστικής που απαιτούνται για τον υπολογισμό των απαντήσεων ενός ατόμου και 2) το πλήθος των αποτελεσμάτων. Τα κόστη αυτά στηρίζονται σε πληροφορία σχετική με τα στιγμιότυπα εννοιών και ρόλων της υπό εξέταση οντολογίας η οποία εξάγεται από ένα αρχικό μοντέλο που κατασκευάζεται από μια OWL μηχανή συλλογιστικής. Πιο συγκεκριμένα, από ένα τέτοιο μοντέλο μπορούμε να εξάγουμε πληροφορία σχετική με το ποια (μη)στιγμιότυπα εννοιών ή ρόλων είναι εύκολο να υπολογιστούν (γνωστά (μη)στιγμιότυπα) και ποια είναι πιο δύσκολο να υπολογιστούν (πιθανά (μη)στιγμιότυπα) την οποία στη συνέχεια χρησιμοποιούμε για τον υπολογισμό των συναρτήσεων κόστους.

Παρουσιάζονται ένας στατικός και ένας δυναμικός αλγόριθμος που χρησιμοποιούν αυτά τα κόστη (που υπολογίζονται από μια στατική και μια δυναμική συνάρτηση αντί-

στοιχα) για να βρουν μια βέλτιστη ή σχεδόν βέλτιστη σειρά εκτέλεσης των ατόμων του ερωτήματος. Σύμφωνα με το στατικό αλγόριθμο τα κόστη (και η σειρά εκτέλεσης των ατόμων του υπό εξέταση ερωτήματος) υπολογίζονται αρχικά πριν την έναρξη της εκτέλεσης του ερωτήματος. Προκειμένου να βρούμε το κόστος εκτέλεσης του επόμενου πιθανού ατόμου, η στατική συνάρτηση λαμβάνει υπόψη της τον τύπο του ατόμου και τις μεταβλητές που έχουν δεσμευτεί από προηγούμενα άτομα. Στη δυναμική περίπτωση τα κόστη υπολογίζονται παράλληλα με την εκτέλεση του ερωτήματος λαμβάνοντας υπόψη πληροφορία σχετική με το αποτέλεσμα της εκτέλεσης των ατόμων του ερωτήματος μέχρι τη δεδομένη στιγμή. Έτσι προκειμένου να βρει το κόστος εκτέλεσης του επόμενου πιθανού ατόμου η δυναμική συνάρτηση λαμβάνει υπόψη της τον τύπο του ατόμου και τις τιμές που έχουν αποδοθεί στις δεσμευμένες μεταβλητές από προηγούμενα άτομα στο πλάνο εκτέλεσης. Στην περίπτωση του δυναμικού αλγορίθμου εκμεταλλευόμαστε μια τεχνική συσταδοποίησης ατόμων που επιτρέπει τον υπολογισμό των συναρήσεων κόστους λαμβάνοντας υπόψη ένα δείγμα αντικειμένου από κάθε συστάδα.

Στη συνέχεια αναπτύσσουμε τεχνικές για τη βελτιστοποίηση ερωτημάτων με σύνθετα άτομα (άτομα που έχουν μεταβλητές στη θέση ατομικών εννοιών και ατομικών ρόλων). Οι τεχνικές αυτές εκμεταλλεύονται τις ιεραρχίες εννοιών και ρόλων της υπο εξέταση οντολογίας (θεωρούμε ότι οι έννοιες και οι ρόλοι της οντολογίας έχουν ταξινομηθεί πριν την έναρξη της διαδικασίας απάντησης ερωτημάτων) προκειμένου να μειώσουν το πλήθος των ελέγχων συνέπειας που εκτελούνται για την απάντηση ερωτημάτων. Για το σκοπό αυτό ορίζουμε την έννοια της πολικότητας μεταβλητών εννοιών και ρόλων σε σύνθετα άτομα και αποδεικνύουμε ότι μπορούμε να διατρέχουμε τις ιεραρχίες εννοιών και ρόλων της οντολογίας, στηριζόμενοι στην πολικότητα των μεταβλητών, με τέτοιο τρόπο, ώστε να κλαδεύουμε κάποιες πιθανές απαντήσεις περιορίζοντας με αυτόν τον τρόπο το πλήθος των ακριβών ελέγχων συνέπειας που πρέπει να εκτελεστούν.

Η αποτελεσματικότητα των προτεινόμενων τεχνικών αξιολογείται χρησιμοποιώντας πρότυπα σύνολα ελέγχου. Από την πειραματική αυτή μελέτη προκύπτει ότι ο στατικός αλγόριθμος ταξινόμησης συνήθως υπερτερεί του δυναμικού όταν είναι διαθέσιμες ακριβείς στατιστικές από την αρχή. Αυτό δεν ισχύει, παρ΄ όλα αυτά, όταν οι στατιστικές είναι λιγότερο ακριβείς, για παράδειγμα εξαιτίας μη ντετερμινιστικών διαδικασιών συλλογιστικής. Η απόδοση του συστήματος συνήθως βελτιώνεται όταν χρησιμοποιούμε τεχνικές συσταδοποίησης κατά τη διάρκεια εκτέλεσης ερωτημάτων. Για σύνθετα ερωτήματα παρατηρούμε μια βελτίωση τάξης μεγέθους τρία όταν χρησιμοποιούνται οι προτεινόμενες βελτιστοποιήσεις. Το υλοποιημένο σύστημα σε συνδυασμό με ένα σύστημα επανεγγραφής ερωτημάτων χρησιμοποιείται στη συνέχεια για την απάντηση ερωτημάτων σε μια πραγματική εφαρμογή στο πεδίο των πολιτιστικών δεδομένων.

Λέξεις κλειδιά: SPARQL απάντηση ερωτημάτων, SPARQL-OWL, σύστημα συνεπαγωγής OWL 2 άμεσης σημασιολογίας, βελτιστοποίηση ερωτημάτων