FAKULTÄT FÜR !INFORMATIK

MASTERARBEIT

Integration of Conjunctive Queries over Description Logics into HEX-Programs

Ausgeführt am

Institut für Informationssysteme
Abteilung für Wissensbasierte Systeme

der

Technischen Universität Wien

unter der Anleitung von

O.Univ.Prof. Dipl.-Ing. Dr. techn. Thomas Eiter

sowie der begleitenden Betreuung von

Dipl.-Ing. Dr. techn. Roman Schindlauer

durch

Bakk. techn. Thomas Krennwallner

Wienerbergerstr. 10, 2513 Möllersdorf

28. September 2007

Integration of Conjunctive Queries over Description Logics into HEX-Programs[*]

Thomas Krennwallner

September 28, 2007

[*]This work has been partially supported by the EC NoE REWERSE (IST 506779) and the Austrian Science Fund (FWF) project P17212-N04.

Abstract

We present *cq-programs*, which enhance nonmonotonic description logics (dl-) programs by conjunctive queries (CQ) and union of conjunctive queries (UCQ) over Description Logics (DL) knowledge bases, as well as disjunctive rules. dl-programs had been proposed as a powerful formalism for integrating nonmonotonic logic programming and DL reasoning on a clear semantic basis. The new cq-programs have at least two advantages. First, they offer increased expressivity by allowing general (U)CQs in the body. And second, this combination of rules and ontologies gives rise to strategies for optimizing calls to the DL-reasoner by exploiting (U)CQ facilities of the DL-reasoner. To this end, we discuss some equivalences which can be exploited for program rewriting and present respective algorithms. Experimental results for the cq-program prototype show that this can lead to significant performance improvements. Moreover, the developed optimization methods may be of general interest in the context of hybrid knowledge bases. HEX-programs, which extend answer-set programming (ASP) with higher-order features and provide powerful interfacing to external computation sources, have been demonstrated to be a versatile formalism for extending the ASP paradigm. The cq-program prototype dl-plugin, which will be introduced in this work, has been developed as a plugin for dlvhex, an implementation for HEX-programs. The dl-plugin integrates ASP with description logics knowledge bases by means of external atoms. For this purpose, a partial equivalence between HEX-programs and cq-programs shows that HEX-programs can serve as a host language for our new formalism, provided that only monotonic dl-atoms appear in the cq-program.

Kurzfassung

Wir präsentieren *cq-Programme*, die nichtmonotone description logics (dl-) Programme um conjunctive queries (CQ) und union of conjunctive queries (UCQ) auf Description Logics (DL) Wissensbasen erweitern, sowie um disjunktive Regeln. dl-Programme wurden als ein mächtiger Formalismus zur Integration von nichtmonotonen logischen Programmen und DL reasonings auf einer klaren semantischen Basis vorgeschlagen. Die neuen cq-Programme haben zumindest zwei Vorzüge. Erstens bieten sie eine gesteigerte Ausdruckskraft aufgrund allgemeiner (U)CQs im Regelrumpf. Und zweitens führt dieser Zusammenschluss von Regeln und Ontologien zu Strategien für die Optimierung von Anfragen an den DL-reasoner durch Ausnützung von (U)CQs des DL-reasoner. Zu diesem Zwecke werden wir Äquivalenzen erörtern, die für Programmumformulierungen ausgenutzt werden und stellen die dazugehörigen Algorithmen vor. Experimentelle Resultate eines cq-program Prototyps zeigen, dass dies zu einer signifikanten Performanzsteigerung führen kann. Darüber hinaus könnten die Optimierungsmethoden von allgemeinem Interesse im Kontext von wiederholten Anfragen auf DL Wissensbasen sein. HEX-Programme, die die Antwortmengenprogrammierung (ASP) um höher-stufige Eigenschaften erweitern und leistungsstarke Schnittstellen für externe Berechnungsquellen bereitstellen, haben sich als ein vielseitig verwendbarer Formalismus zur Erweiterung des ASP Paradigmas bewährt. Der cq-Programm Prototyp dl-plugin, der in dieser Arbeit vorgestellt werden wird, wurde als Plugin für dlvhex, einer Implementation für HEX-Programme, entwickelt. Er integriert ASP mit Beschreibungslogik-Wissensbasen anhand von externen Atomen. Eine partielle Äquivalenz zwischen HEX-Programmen und cq-Programmen zeigt, dass HEX-Programme als Host-Sprache für unseren neuen Formalismus fungieren kann, solange nur monotone dl-Atome im cq-Programm vorkommen.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

First and foremost, I would like to give my best thanks to my supervisor, Thomas Eiter, for giving me the opportunity to work on this very interesting topic, for all his efforts, and our numerous meetings, where we settled the foundations of this work. In these sessions, I learned uncountable many new insights. Moreover, I am very grateful for the student grant I received and the possibility to visit the International Workshop on Description Logics 2007 in Brixen/Bressanone, Italy, and the Summer School Reasoning Web 2007 in Dresden, Germany.

I express my deepest gratitude to my fellow co-authors Giovambattista Ianni and Roman Schindlauer for all their help, proofreading, and valuable comments. This work started with a student project supervised by Roman Schindlauer, hence I am obliged for the persistent collaboration in implementation details.

Moreover, I say thank you to my fellow students Michael Jakl and Erik Gostischa-Franta, who did not surrender reading early drafts of this thesis. I would also like to thank my close friends from the "graphics world," Stef Grimm, Yasmin Mehra, and Thomas Weniger, who helped me with valuable explanations and hints on the process of image processing, desktop publishing, and professional printing for the DL'07 and the epilog poster. In my view, this is a highly non-trivial task.

I am also pleased to thank Elfriede Nedoma, for her efficient work on planning travel arrangements and other administrational work. Moreover, big thanks goes to the whole faculty of KBS, since I enjoined all of them in lectures.

And I would like to thank my family for making my studies in Computational Intelligence possible. This goes in line with the caption from [Tennant, 2004], which shows a father approaching his son:

> *"It still bothers me that I'm paying a lot of* real *dollars to a* real *university so you can get a degree in* artificial *intelligence."*

Kirk:	Another technical journal, Scotty?
Scotty:	Aye.
Kirk:	Don't you ever relax?
Scotty:	I am relaxing.

—Star Trek, *The Trouble With Tribbles*, stardate 4523.3

1 Introduction

In this thesis we cover a novel formalism for combining rules and ontologies for the Semantic Web, called cq-programs. We will present implementation details and its optimization features with experimental data. This work belongs to the wide field of knowledge representation and reasoning (KRR), which adheres closely to Artificial Intelligence (AI). cq-programs build on the treatise of other combinations of rules and ontologies, the so-called dl-programs [Eiter et al., 2004b,c, 2007b], and on HEX-programs [Eiter et al., 2005b, 2006b]. A more detailed elaboration on both languages is subject to [Schindlauer, 2006].

The central theme of this thesis is to investigate a state-of-the-art combination of a rule formalism with ontologies for the Semantic Web. The Semantic Web—being introduced by Tim Berners-Lee, the inventor of the World Wide Web, in the early 1990s—delivers the means for describing semantic relationships and the data for building an enormous quantity of worldwide easily accessible knowledge bases. The remaining gap for intrinsic limitations of ontology languages is filled by rule formalisms (see more in Section 1.3 and [Rosati, 2006b, Motik et al., 2006, Eiter et al., 2006a]). The rule formalism in our setting is answer-set programming with special kinds of query atoms in the bodies of rules. Together, both KRR artifacts will provide the key to the evolvement of the vision behind the Semantic Web. This goes in line with the constitutive architecture of the Semantic Web, shown in the Semantic Web architecture (Figure 1.3).

The main contributions, briefly summarized, are as follows:

1. We present the cq-program formalism.
2. We show a partial equivalence between cq-programs and HEX-programs.
3. We report on the implementation of cq-programs as a plugin for dlvhex, a HEX-program solver.
4. Rewriting rules for cq-programs establish valuable optimization means.
5. Experimental results show the virtue of optimizing cq-programs with respect to evaluation time.

The upcoming sections will give more hints on the underlying machinery used in this work. Furthermore, we set the ideas for the underpinning of cq-programs and its implementation, the dl-plugin. This concludes the big picture behind this thesis.

The results on cq-programs and its optimization feature had been published in the *Proceedings of the 20th International Workshop on Description Logics DL'07* [Eiter et al., 2007a] and the dl-plugin implementation was presented in the demonstration session of this workshop.

1.1 Answer-Set Programming

Logical programming is used in knowledge representation since the early 1970s [Kowalski, 1974]. *Prolog* (Programming in Logic) was the first programming language which used mathematical logic for knowledge-based problem solving (for a historical account on Prolog, see the article [Colmerauer and Roussel, 1993]). In [Kowalski, 1979], the formula "Algorithm = Logic + Control" was proposed to promote the idea that an algorithm is composed of a logical component, which specifies the knowledge respecting a certain problem domain, and that of a control unit, which decides the execution of the knowledge in the logic component. With this idea in mind, the strict separation of both pieces should support programming in general, purely and simply because the evaluation of programs in the control unit can deal with efficiency criteria without touching the logic component. This opportunity will also be exploited in our framework. A historic overview of logic programming is presented in [Apt, 1990].

For an inaugural example, take the statements "every human is mortal" and "Sokrates is human." The two of them are represented in a logic programming language such as Prolog by the fact

$$human(sokrates)$$

and the rule

$$mortal(X) \leftarrow human(X).$$

From this formalized knowledge, we deduce the fact $mortal(sokrates)$, that is, Sokrates is a mortal being. In such programs, we have atoms of the form $human(X)$, and $human$ is a predicate name. Note that we only specify the knowledge of our problem-solving domain, the process of deduction is encoded in the particular Prolog inference engine.

The Logic Programming paradigm has since then been applied in many domains, from Production Systems to Deductive Databases. One representative is *Answer-Set Programming* (ASP) introduced in [Gelfond and Lifschitz, 1991]. The ASP approach to logic programming differs from traditional logic programming in the following ways. First, it has pure declarative semantics, i.e., in contrast to Prolog programs, the order of rules in a program does not matter. Secondly, solutions to problems encoded in answer-set programs are determined by the models of such programs. And thirdly, it uses both strong and weak negation, where the latter is also known as *Negation As Failure* (NAF). Due to NAF, the semantics for answer-set programs are nonmonotonic, i.e., the set of logical consequences might decrease with increasing information in the program. On that account, ASP is a convenient approach for expressing incomplete and inconsistent information in logic programs.

To give an illustrative example for an answer-set program, we show a program encoding for the graph three-colorability problem (3COL). The 3COL problem is the following: given a graph $G = (V, E)$, is it possible that we can assign a color $c \in \{red, green, blue\}$ to every vertex $v \in V$, such that for every edge $(v_1, v_2) \in E$ the vertices v_1 and v_2 have a different assigned color? This problem is a well-known NP-complete problem (cf. for instance [Garey and Johnson, 1979]).

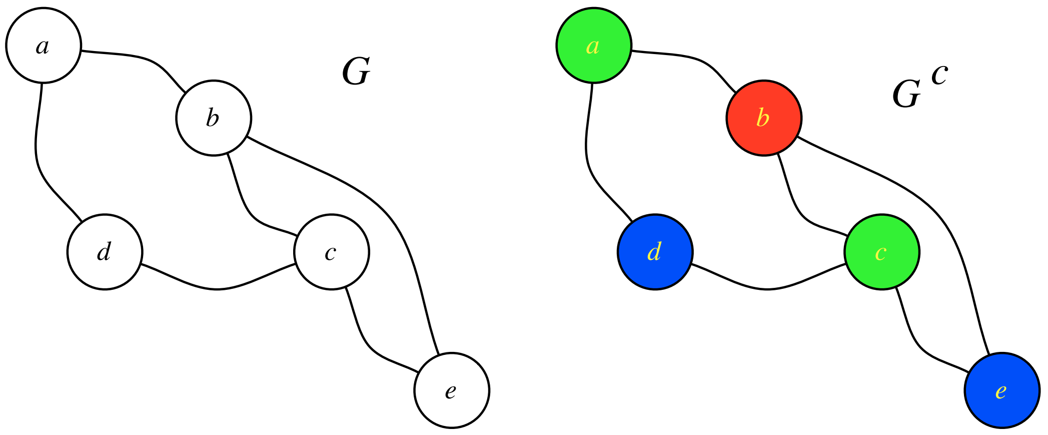

Figure 1.1: A three-colorable graph

For a problem instance we use the graph G shown in Figure 1.1; it is easy to see that G is three-colorable, one particular solution is the graph G^c. The next step is to encode the problem instance G as answer-set program P, such that G is three-colorable iff P has an answer set.

$$\begin{array}{rl}
\mathit{col}(X,\mathit{red}) \lor \mathit{col}(X,\mathit{green}) \lor \mathit{col}(X,\mathit{blue}) \leftarrow \mathit{node}(X). & \big\}\ \text{guess solution}\\
\leftarrow \mathit{col}(X,C), \mathit{col}(Y,C), \mathit{edge}(X,Y). & \big\}\ \text{check solution}\\
\begin{array}{l}\mathit{node}(X) \leftarrow \mathit{edge}(X,Y).\quad \mathit{node}(X) \leftarrow \mathit{edge}(Y,X).\\ \mathit{edge}(a,b).\ \mathit{edge}(b,c).\ \mathit{edge}(c,d).\ \mathit{edge}(d,a).\ \mathit{edge}(e,c).\ \mathit{edge}(e,b).\end{array} & \Big\}\ \text{graph encoding}
\end{array}$$

Without going into detail on the formal semantics of an answer-set program we intuitively describe what P does. Basically, P is divided into three parts: a guessing part, which guesses a coloring of the nodes, a checking part, which checks if the guessed solution is a valid three-coloring, and a graph encoding, which defines the edges and nodes of the problem instance. We encode the binary relation E as atoms with the predicate *edge*, the set V of nodes as unary atoms with the predicate *node*, and each colored node is represented as a binary atom $\mathit{col}(N, C)$, where N is a node and C is one of the colors *red*, *green*, and *blue*. The graph encoding part is the easiest part to grasp, we just provide the edge relation E as facts, while V is deduced from the edges by the rules $\mathit{node}(X) \leftarrow \mathit{edge}(X, Y)$ and $\mathit{node}(X) \leftarrow \mathit{edge}(Y, X)$, which intuitively says that for every $(v_1, v_2) \in E$, v_1 and v_2 are nodes in V. In the guessing part, we "guess" a solution from each node $\mathit{node}(X)$ to colored nodes $\mathit{col}(X, \mathit{red})$, $\mathit{col}(X, \mathit{green})$, and $\mathit{col}(X, \mathit{blue})$. The checking part "kills" solutions, which have equally colored connected nodes.

Note that this simple program not only solves our decision problem, whether G is three-colorable, it provides the solution encoded in its answer sets as well. In fact, G is three-colorable and there are 18 different three-colorings for this graph, hence P has 18 answer sets, one for each solution. The solution G^c holds the corresponding digested answer set $\{\mathit{col}(a, \mathit{green}), \mathit{col}(b, \mathit{red}), \mathit{col}(c, \mathit{green}), \mathit{col}(d, \mathit{blue}), \mathit{col}(e, \mathit{blue}), \dots\}$ of P.

Furthermore, the ASP formalism has been proven to be a suitable approach for acting as a host language for encoding advanced reasoning tasks. In the DLV system, frontends for Planning [Eiter et al., 2001] (in particular the action language $\mathtt{DLV}^{\mathcal{K}}$) and Diagnosis [Eiter et al., 1999] tasks have been implemented. The dlvhex system adds support for *higher-order logic programs* (which accommodate meta-reasoning through higher-order atoms) and *external atoms* for software interoperability. One can think of external atoms as a *foreign* *function interface* for accessing services provided by other programming languages or reasoning facilities; this also gives hints on the implementation of cq-programs. Since dlvhex is such a versatile system, we make use of it in our dl-plugin.

1.2 The Semantic Web

In this section, we introduce the most important aspects of the Semantic Web by some historical facts and describe the inventions made during the portrayed period of time.

The *World Wide Web* (or WWW for short), as we know it today, is an offspring of CERN. This famous European Particle Physics Laboratory located in Geneva/Switzerland hosts the laboratory equipment for thousands of researchers from all over the world. As a consequence, this stimulated a need for *structured archiving and accessing* of research data, experimental results, project proposals, telephone numbers, and similar information. Tim Berners-Lee, back in the 1980s a fellow at CERN, was the prime inventor of the protocols and tailored the programs for his vision of a *Web of Documents and Data*. Eventually, the first web server was brought up at CERN on August 6th 1991. What follows, is the dramatical success of the WWW.

The basic idea behind the Web is *hypertext*, i.e., text combined with references to other information. Such information can be on the very same document or stored in other documents on the Web, like another hypertext, or an image, and so on. Since every document on the Web may link to other Web resources, a standardized framework has to come up to enable compatible and interchangeable programs to share its data. Therefore, in 1994, the *World Wide Web Consortium* (W3C)[1] started its standardization work. Fundamental standards for the Web emerged, such as *Hypertext Transfer Protocol* (HTTP) for transferring arbitrary data over networks, *Uniform Resource Identifier* (URI) to link Web resources, and *Hypertext Markup Language* (HTML) for representing hypertext with URIs. With these standards, the successful story of the WWW was grounded and is nowadays a central resource for information gathering.

Later in 1998, another groundbreaking standard came into light. Due to the substantial amount of information available on the Web, mostly in unstructured form, the first standard for a general markup language, the *Extensible Markup Language* (XML),[2] was issued from the W3C. The semi-structured data-model [Abiteboul, 1997] is the basis for XML and allows for specifying well-formed documents on the Web. Since XML was designed to be a general markup language, everyone is now able to create custom XML-based languages. The XML framework provides the means for describing schematic information about XML documents and thus offers the ability to automatically check the validity of XML documents. An example for an XML document is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book> <author>A</author> <name>B1</name> </book>
  <book> <author>C</author> <name>B2</name> </book>
</books>
```

books	
author	*name*
A	B1
C	B2

This XML document encodes information for the books B1 and B2 from authors A and C, resp. The table on the right shows the relational presentation of the XML document.

Moreover, by means of XML Namespaces, we can prefix specific elements in an XML document with an identifier to facilitate interoperability with other XML-based languages; the RDF example below makes heavy use of this concept.

[1] `http://www.w3.org/`

[2] `http://www.w3.org/XML/`

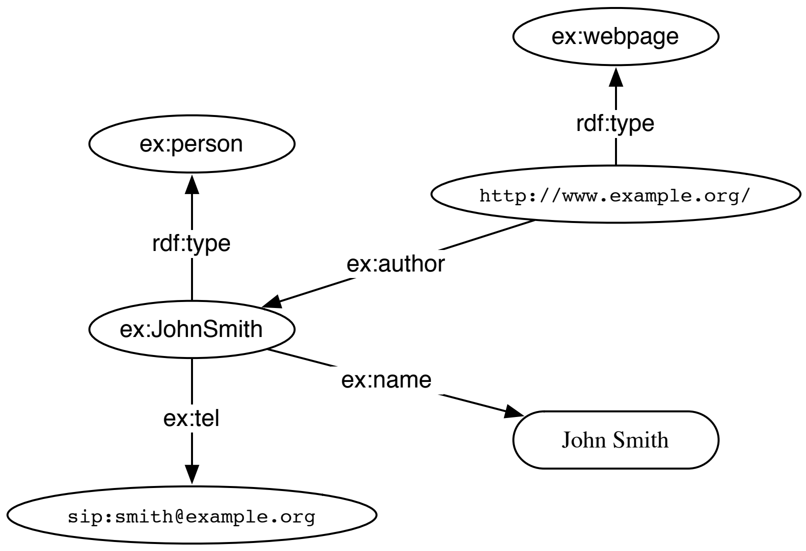

Figure 1.2: An RDF Graph

Given that documents on the Web are henceforth free to be represented in a structured manner, XML-based languages are another big stepping stone on the way to publishing data on the Web. In the article [Halevy et al., 2006], the authors argue that the development of XML has been a great success in order to provide a common framework for sharing data among data sources; one can view XML as a driving force for the realm of *Data Integration* (see for instance [Halevy, 2001]).

A year later, the *Resource Description Framework* (RDF)[3] specification was proposed at W3C. As stated in [Manola and Miller, 2004], this language is an appropriate formalism for representing resources on the Web and was mainly developed for automatic data processing of meta-information. Using the RDF language, statements such as

`http://www.example.org/` has an author named John Smith

are easily expressible. With such information at hand, one can deduce facts about things described in RDF. The prior statement also shows the basic idea behind RDF, namely that the data has assigned properties, which results in expressing such meta-data in a graph-like manner. Speaking in RDF terms, our statement has a *subject* `http://www.example.org/`, a *predicate* author, and an *object* John Smith. With the help of XML, we describe the above statement (with some additional information) as a RDF/XML representation:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:ex="http://www.example.org/ns/"
   xml:base="http://www.example.org/ns">
  <ex:webpage rdf:about="http://www.example.org/">
    <ex:author rdf:resource="#JohnSmith" />
  </ex:webpage>
  <ex:person rdf:ID="JohnSmith">
    <ex:name>John Smith</ex:name>
    <ex:tel rdf:resource="sip:smith@example.org" />
```

[3] `http://www.w3.org/RDF/`

```
  </ex:person>
</rdf:RDF>
```

The corresponding RDF graph of the foregoing RDF document is shown in Figure 1.2.

In the year 2000, the book [Berners-Lee and Fischetti, 2000] and the article [Berners-Lee et al., 2001] gave the intuition behind the *Semantic Web*. In 2001, the *W3C Semantic Web Activity*[4] came into existence, and is since then a starting point for all things related to the Semantic Web. The Web with all its content was mature and well-established among Internet users, but what was missing according to Tim Berners-Lee, was a *Web of Data*, i.e., the means for machine-readable information and automatic data processing, ultimately leading to more powerful usage of data. For this to succeed, computers should not only support the user when searching information on the Web, but should even "understand" what the data and a given query "means." The data published on the Web must contain semantical information in form of terminological knowledge such as ontologies, which will be addressed later on. This will enable much more power- and meaningful information processing on the Web, including data discovery and integration, and it will improve automation of tasks. To put it simply, one intention behind the Semantic Web is to delegate time-consuming and tedious work such as repetitive Web searching usually done by the user herself to machines.

To accomplish these objectives, the *Semantic Web Architecture* is split into hierarchical parts, with each level assigned a particular task. According to Berners-Lee [1998], and following Hendler [2002], the Semantic Web is divided into the following layers: *URI + Unicode*, *XML*, *RDF + RDF Schema*, *OWL*, *Logic*, *Proof*, *Trust*, and *Digital Signature*.

- The bottom layers consists of foundational frameworks for identifying Web resources (*URI*) and for representing text in a uniform way over different computer platforms and natural languages (*Unicode*);
- on the next level, the *XML* framework is used for annotating information to assist data sharing;
- the third layer deals with expressing meta-data about Web resources with the help of *RDF* and its extension *RDF Schema*;
- the fourth layer embraces ontology vocabularies like *OWL* (see the upcoming discussion) to express terminological knowledge and assign semantics to concepts;
- the final layers deal with *Logic*, *Proof*, and *Trust* issues. Similar to the *Trust* layer, the *Digital Signature* component is supposed to manage cryptographic technologies for assuring identification issues of Web resources.

Recent updates on the Semantic Web architecture (see slides from [Herman, 2007]) incorporate changes to a more refined version with respect to proposed and upcoming Web standards and current state-of-the-art research on these topics (see Figure 1.3). In this version of the Semantic Web architecture, we see how *Rules*—specified in the upcoming W3C standard *Rule Interchange Format* (RIF)[5]—fit into the big picture. Since many different rule formalisms exit today with different proposed syntax and semantics, this attempt tries to single out the common features of rule languages to build a uniform markup language with a proper model-theoretic semantics for exchanging rules in—and not exclusively limited to—the Semantic Web context. The *Query* part consists of the *SPARQL* query language for RDF,[6] which will soon be proposed as a W3C standard.

[4]`http://www.w3.org/2001/sw/`
[5]`http://www.w3.org/TR/rif-core/`
[6]`http://www.w3.org/TR/rdf-sparql-query/`

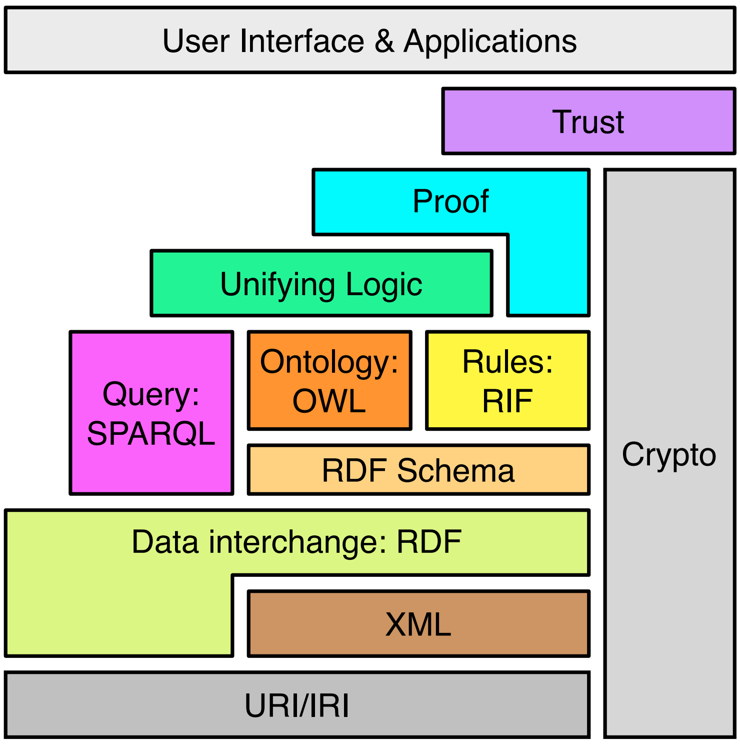

Figure 1.3: The Semantic Web Architecture

Unifying Logic is a current hot topic in the research community; this part should provide combined formalisms for ontologies and rules by embedding the two of them to different logical formalisms (cf. Section 1.3). Moreover, an additional piece called *User Interface & Applications* shows how Semantic Web applications will interface with other parts of the architecture. See [Horrocks et al., 2005] and [Patel-Schneider, 2005] for an account on various proposed architectures in the past and pictures thereof.

In 2004, the next milestone towards the Semantic Web had been reached. The ontology vocabulary called *Web Ontology Language* (OWL) [Patel-Schneider et al., 2004, McGuinness and van Harmelen, 2004],[7] which builds syntactically upon RDF and RDF Schema, was proposed as a standard to define ontological knowledge in Semantic Web applications. With this, one may express complex relationships between classes and properties, and assert membership information of individuals in such ontologies. To clarify these points, take as example a family hierarchy, i.e., the relationships that can occur in family-related terminological knowledge, such as “a Mother is a Woman with at least one Child,” “a Son is a Man with a Parent,” or “John is the Son of Mary.” To express such knowledge we have to define classes for Woman and Man, and the usual disjoint relationship among such classes. Then, some women are mothers, which is expressed by defining that the property of hasChild has positive cardinality over the domain Woman. Similarly, a Man which hasParent property has positive cardinality is a Son. A visual representation of this family hierarchy is shown in Figure 1.4. The possibility for expressing such information is crucial in the Semantic Web, since we are now fit to reason over the knowledge expressed in our family ontology and ask questions—for instance “is Mary a Mother?”—, whose answers will be derivable in an automatic fashion. OWL is divided in three increasingly expressive languages called OWL Lite, OWL DL, and OWL Full, where OWL DL and OWL Lite is

[7]`http://www.w3.org/2004/OWL/`

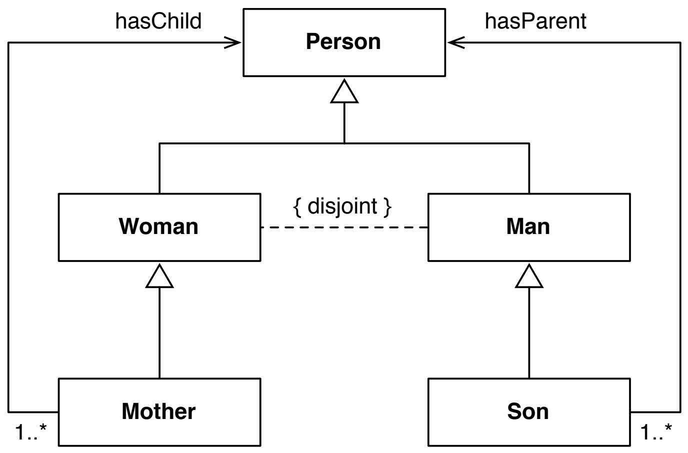

Figure 1.4: A family hierarchy

influenced by *Description Logics* (DL), a family of logics for formalising such terminological information and with well-studied reasoning services. In particular, OWL DL and OWL Light are based upon the DLs $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SHIF}(\mathbf{D})$, resp. Our family hierarchy may be equivalently and compactly represented by the following DL axioms:

$$\begin{array}{rlr} Woman \sqsubseteq Person & Woman \sqsubseteq \neg Man & Man \sqsubseteq Person \\ Mother \sqsubseteq Woman & & Son \sqsubseteq Man \\ \exists hasChild.Person \sqsubseteq Mother & & \exists hasParent.Person \sqsubseteq Son \end{array}$$

Without any details, the $\sqsubseteq$ operator can be imagined as an `is-a` relationship, for instance $Woman \sqsubseteq \neg Man$ states that "a Woman is not a Man." One can easily think of how such logical statements are represented in an XML-based ontology language such as OWL. See [Horrocks et al., 2003] for more details on the connection between OWL and description logics.

Given the constituents of the Semantic Web architecture shown in Figure 1.3, what has not been agreed on are the final parts not yet mentioned in our historic overview. Therefore, in 2004, the *RDF Data Access Working Group*[8] was formed for creating standardization work for the SPARQL query language for RDF, which has been released as a Candidate Recommendation recently. In 2005, the *Rule Interchange Format Working Group*[9] was formed to devise standards for interchanging rules on the Semantic Web.

Recently, a new revision of the OWL DL Ontology Language, which is provisionally called OWL 1.1,[10] has been proposed [Cuenca Grau et al., 2006] and was accepted as a W3C Submission. This extended OWL formalism is based on the DL $\mathcal{SROIQ}$.

This concludes our overview of the Semantic Web. The upcoming section deals with issues that arise when combining rules with ontologies. What started with the basic building blocks has now been refined with more details. Since the *Rules* component will play a crucial role in the higher layers such as *Proof*, *Trust*, and *Unifying Logic*, we must accept that without a properly defined and expressive rule part the vision of the Semantic Web cannot come true. As a consequence, rule languages must interface with ontologies. Berners-Lee et al. [2001] write in their Scientific American essay:

[8]`http://www.w3.org/2001/sw/DataAccess/`
[9]`http://www.w3.org/2005/rules/wg`
[10]`http://webont.org/owl/1.1/`

"The challenge of the Semantic Web, therefore, is to provide a language that expresses both data and rules for reasoning about the data and that allows rules from any existing knowledge-representation system to be exported onto the Web. Adding logic to the Web—the means to use rules to make inferences, choose courses of actions and answer questions—is the task before the Semantic Web community at the moment."

1.3 Integration of Rules and Ontologies

In this section, we present certain limitations of ontology formalism, shed light on issues that may arise when integrating rules and ontologies, and survey existing formalisms in this context.

1.3.1 Motivation

During research on Semantic Web technologies the demand for combined formalisms, which integrate ontology and rule languages, emerged as a consequence to supply advanced reasoning capabilities in this setup. Ontology languages on their own cannot fulfill all enjoined requirements; rule languages like logic programs should close at least some of the known obstacles.

In this thesis, we focus on the answer-set programming way to logic programming. As will be shown subsequently, this approach is common in related work on combined rule/ontology languages for Semantic Web reasoning.

As for now, several approaches for the combination of rules and ontologies exist. One exponent in this direction is the *Rule Markup Language* (RuleML) [Boley et al., 2001], which is an XML-based language for expressing rules. Another example for ontology languages with integrated rule part is the *Semantic Web Rule Language* (SWRL) [Horrocks et al., 2004]. Alternative contributions in this field exist, mostly based on integrating DLs with logic programs. We will give more examples of such combinations in this section.

Description logics are the fundamental underpinning for many ontology languages [Baader et al., 2005b], thus we focus hereby on DLs. Following the list of motivations in [Motik et al., 2006], we give some known expressive limitations of DL formalisms that show the need for combining logic programming and DLs. Another relevant work in the context of incomplete information—the assertional knowledge in a DL knowledge bases is conceivable as incomplete database—, which provides more in-depth discourse, can be found in [van der Meyden, 1998].

Higher Relational Expressivity The DLs underlying the ontology languages OWL DL and OWL Light can only model domains with objects connected in a tree-like shape (see [Vardi, 1997] for a characterisation of the tree-model property). This restriction is annoying due to the fact that many real-world applications have a need to express relational structures in a triangle form, for instance the expression that "an uncle is the brother of a person, with both having the same father" is not expressible in such DLs. But the preceding statement is very easy to express in a rule language, as the following rule shows:

$$\mathit{uncle}(X) \leftarrow \mathit{brother}(X,Y), \mathit{fatherOf}(Z,X), \mathit{fatherOf}(Z,Y).$$

The discussion in [Motik et al., 2004] provides a more detailed view on this issue. Observe that the upcoming OWL 1.1 standard supports representing such triangular structures by virtue of complex role inclusion axioms of form $R \circ S \sqsubseteq T$.

Polyadic Predicates The building blocks of DLs are concept and role expressions, which can be seen as unary and binary predicates in a first-order theory. Hence, relationships with higher arities cannot be expressed immediately, one needs to apply the reification technique to simulate n-ary relations in DLs (cf. [Baader et al., 2003]). Logic programs, on the other hand, are naturally equipped with n-ary predicates. An example for a predicate of arity 5 would be a train schedule entry like

$$\mathit{train}(\mathit{ice562}, \mathit{vie}, \mathit{szg}, \text{“6:14”}, \text{“8:53”}),$$

which specifies that the train ICE 562 leaves at 6:14 from Vienna Western Station and arrives at 8:53 in Salzburg Central Station; such a predicate is not expressible in OWL. There are DLs with n-ary roles, take $\mathcal{DLR}_{\mathit{reg}}$ in [Calvanese et al., 2007b], but they are not under consideration in ontology languages used in the Semantic Web setting. Using reification, we would have to introduce five fresh roles $R_1, \ldots, R_5$ and associate a new concept *Train* with each of these roles by the concept inclusion axiom[11]

$$\begin{aligned}\mathit{Train} \sqsubseteq{} & \exists R_1.\mathit{TrainID} \sqcap \exists R_2.\mathit{Station} \sqcap \exists R_3.\mathit{Station} \sqcap \exists R_4.\mathit{Time} \sqcap \exists R_5.\mathit{Time} \sqcap \\ & \leq 1 R_1 \sqcap \cdots \sqcap \leq 1 R_5,\end{aligned}$$

provided that the first argument of predicate *train* is of type *TrainID*, the second and third arguments are *Station* members, and the last two arguments are *Time* members (alternatively, we could use a datatype, for instance xsd:time). We will now encode our predicate *train* with the next ABox axioms, where the fresh individual t_1 identifies our *train* tuple:

$$\mathit{Train}(t_1); \quad R_1(t_1, \mathit{ice562}); \quad R_2(t_1, \mathit{vie}); \quad R_3(t_1, \mathit{szg}); \quad R_4(t_1, \text{“6:14”}); \quad R_5(t_1, \text{“8:53”}).$$

Compared to the simple 5-ary predicate *train*, the reified DL-version of such a train schedule is very cumbersome.

Integrity Constraints Integrity constraints, while extensively used in relational database applications, are not expressible in DLs. In fact, [Reiter, 1988] shows that integrity constraints are not expressible using first-order sentences. Integrity of an ontology should be enforced by modal nonmonotonic logic or a rule language. See [Motik et al., 2007a,b] for an account on this issue and for a proposed solution, which uses so-called *extended* DL knowledge bases. Such extended DL-KBs consist of an additional set of constraints in the ontology, which are, together with the ABox assertions, interpreted under a minimal model semantics. However, many rule languages support expressing constraints by design, thus reusing them would be compelling. Moreover, extended DL-KBs cannot access the rules layer, hence such constraints would be limited to ontologies. An example for such a constraint is the rule

$$\leftarrow \mathit{col}(X, C), \mathit{col}(Y, C), \mathit{edge}(X, Y)$$

in our 3COL example in Section 1.1, which forces the answer sets of this program to valid three-colorings of a graph.

Modeling Exceptions Support for exceptions is not available in DLs. Take, for instance, the well-known Tweety example [Reiter, 1978b]. As taught in primary school, everyone knows that penguins are birds. Usually, birds fly, but exceptionally, penguins cannot fly.

[11]Proper use of reification would introduce key constraints. We ignore this here, hence multiple members of *Train* might identify a certain *train* tuple.

Suppose now that Tweety is a bird, therefore we infer that she can fly. But if we know that Tweety is a penguin, we infer that Tweety is able to fly and unable to fly at the same time, which is clearly inconsistent. Such problems show the need for modeling exceptions. In fact, one of the driving forces for nonmonotonic formalisms such as answer-set programming is the possibility to model such exceptions; the NAF operator provokes this exceptional behaviour in ASP and is thus used as a handy tool to express exceptions.

1.3.2 Issues

We address now the problems and some crucial design issues that may arise in such combination efforts. The discussions in [Rosati, 2005b, 2006b] raise some matter of importance in this respect. Additionally, [Bry and Marchiori, 2005] give theses on kinds of logics needed for the Semantic Web, and [de Bruijn et al., 2006] provide principles on the representational issues in such combinations.

OWA vs. CWA The *Open World Assumption* (OWA) is adopted in first-order logic and their fragments such as description logics. OWA maintains an agnostic view of the world, i.e., conclusions which cannot be derived from a first-order theory are not known to hold and would not be taken into account as “result,” so it is possible that some facts may hold in some problem domain which cannot be deduced, but it is open for which facts.

In contrast, the *Closed World Assumption* (CWA) maintains, hence the name, a closed view of the world [Reiter, 1978a]. Everything which is not derivable from a CWA knowledge base is assumed to be false. Take, for instance, a consumer database recording the customers of a shop. Such a database is assumed to be complete in a sense, that for every individual not listed in this database one deduces she has not been buying products at that shop. This is indeed a reasonable assumption, since otherwise we would have to catalogue all persons not buying anything at that shop. CWA is closely related to negation as failure in logic programming, since for every fact we fail to derive a proof in a CWA theory, we can assume that the negation of it holds.

However, in the context of the Semantic Web, the CWA is not adopted in ontologies. This is due to the fact that (i) ontologies are based on description logics and (ii) the “state of knowledge” on the Web is to a certain extent in constant flux; it would be highly devastating if one conclusion found at a point in time must be invalidated at a future time. This is closely related to the Proof layer of the Semantic Web architecture. See also [Patel-Schneider and Horrocks, 2006] for a more detailed account.

But nonmonotonic reasoning, which uses CWA, is needed in many applications. Take our consumer database as one example for the relevance of such sort of reasoning. Integrating a nonmonotonic formalism with ontologies is therefore a key requirement. Since logic programming knowledge bases are kept under the CWA, the interaction between either semantic viewpoint in combined rule and DL systems is inherent, the “proper” way of doing this is still unknown; as shown later on, many different proposed formalisms exist today.

UNA vs. non-UNA The *Unique Name Assumption* (UNA) requires that two individuals mentioned in the problem domain are denoted by two different individual names. This is the standard semantics for rule languages, but UNA is not adopted for some DLs, and in particular not in the DLs used in OWL DL and OWL Light. OWL has the construct `owl:sameAs`, which is suitable to specify that two individual names denote the same individual. Again, the integration of rules and ontologies may lead to troubles in this respect.

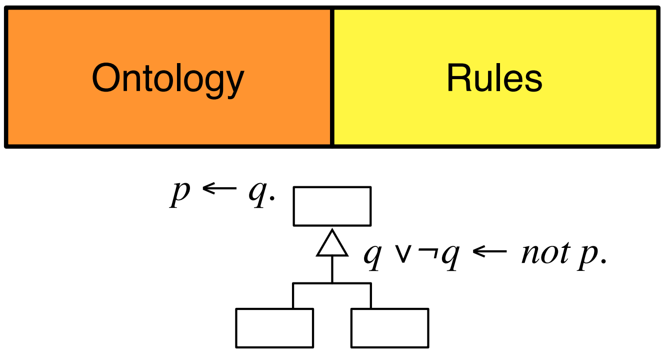

Figure 1.5: Tight Coupling

Decidability Preservation In one of the first integration efforts for rules and ontologies dubbed CARIN [Levy and Rousset, 1998], it was shown that this coalition quickly leads to an undecidable reasoning mechanism, even if reasoning in the logic program and the DL on their own is decidable. See also the discussion in [Motik et al., 2004] for the reasons causing undecidability of the reasoning procedure in combined knowledge bases.

Modularity of Reasoning Reasoning in DLs and logic programs had been heavily studied in the past and many implementations are available. Thus, it would be alluring when combining rules and ontologies to use currently available and mature software components. As we will see, other integrated formalisms have been proposed in the literature, and some come with an implementation. From the conceptual point of view they all require a new implementation from scratch due to their tight semantic integration, whereas our approach differs in this respect; we are free to reuse existing components without touching the internals too much.

Continuing the discussion about modularity of reasoning, [Eiter et al., 2006a] proposes two strategies for creating combined knowledge-based systems. In principal, the integration efforts are roughly categorized in a

1. *tight semantic integration* or *tight coupling* (see Figure 1.5), and in a
2. *strict semantic separation* or *loose coupling* between the DL and the rule system (see Figure 1.6).

In view of the interaction between combined knowledge bases with a classical and a nonmonotonic part in [Eiter et al., 2006a]—the mentioned systems in the upcoming rest of this section are basically such combined KBs—, tight coupling is generalized as Principle 3.1 (Interaction based on single models), while the loose coupling is stated as Principle 3.2 (Interaction based on entailment) in this work. We will now review systems which are based on these interaction principles.

Tight semantic integration In this approach, the rule semantics are adjusted to fit into the ontology layer. As pointed out previously in this section, such a combination can quickly lead to an undecidable reasoning mechanism in case of CARIN [Levy and Rousset, 1998] and SWRL [Horrocks et al., 2004]. On the other end of the scale, the decidable approach called DLP [Grosof et al., 2003] is very restrictive, i.e., it only allows a fusion of logic programming and DLs in the semantic intersection of both concepts, which is limited in the expressivity.

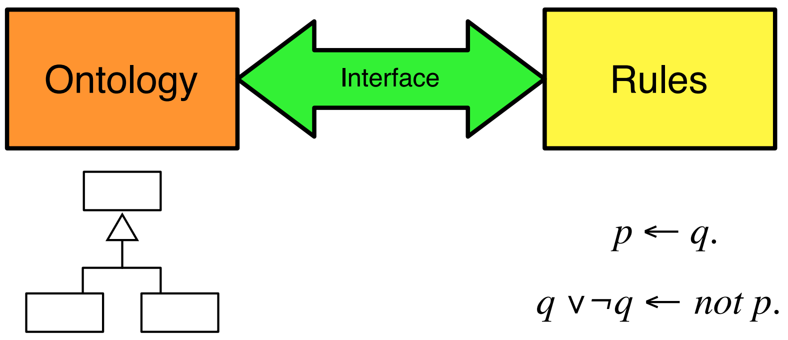

Figure 1.6: Loose Coupling

Several other approaches have been studied, which try to close the gap between undecidable formalisms and the restrictive DLP proposal, mainly in the area of ASP. All these approaches straiten the integration of rules and ontologies by adding a safety restriction to the rules in such combinations, for instance, variables in rules can only occur in certain places. Approaches such as $\mathcal{DL}$-log [Rosati, 1999], DL-safe Rules [Motik et al., 2005], safe hybrid KBs [Rosati, 2005b], r-hybrid KBs [Rosati, 2005a], and the expressive $\mathcal{DL}$+log [Rosati, 2006a,b] fall under this category. Recently, [Lukasiewicz, 2007] presented a tightly integrated approach called *disjunctive dl-programs under the answer-set semantics.*

Additionally, an approach based on reducing DL-KBs to disjunctive logic programs (cf. [Hufstadt et al., 2004]) is used in the DL-reasoner KAON2.[12] Here, every $\mathcal{SHIQ}$ knowledge base is transformed into an equivalent disjunctive logic program, and reasoning in the DL is performed using standard reasoning algorithms for such programs. A positive side-effect of this approach is that one can easily add a rule layer to a DL-KB; in fact, this amounts to appending the rules of the logic program to the translated DL-KB—the above-mentioned DL-safe rules are implemented this way.

Moving on, another class of proposed formalisms exists. They can be considered as embeddings of logic programming and first-order theories into unifying formalisms. The first one to mention is the Hybrid MKNF KB formalism [Motik et al., 2006, Motik and Rosati, 2007], which uses the logic *Minimal Knowledge and Negation as Failure* for integrating DLs with logic programming. Another approach uses *Autoepistemic Logic* [de Bruijn et al., 2007a], while [de Bruijn et al., 2007b] uses *Quantified Equilibrium Logic.*

Strict semantic separation This strategy aims at keeping the rules and the ontology strictly apart from each other. The only way an information exchange can happen between both layers is through a “safe interface.” The rule layer views the ontology as an external unit which can be queried and augmented by knowledge from the rule part; the semantics of the ontology and the rule layer is kept independent from each other. This approach is typical for the formalisms proposed in [Eiter et al., 2004b, 2005b, Heymans et al., 2005] and [Lukasiewicz, 2005]. Our approach to combining knowledge bases is such a strict separation (see Chapter 3 for the definition of cq-programs).

For excellent surveys which classify the above-mentioned and other approaches we refer the interested reader to [Antoniou et al., 2005] and [Pan et al., 2004].

The motivation for building integrated rule and ontology knowledge bases clearly follows from the aforementioned issues and the requirements for the Semantic Web architecture.

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

Efforts in this direction are currently a hot topic in the Logic Programming, Description Logics, and Semantic Web research communities. Hence, this thesis may be a valuable contribution to further the results in these fields, especially in the light of more efficient reasoning capabilities for the Semantic Web.

1.4 Thesis Organization

We now outline this thesis. Related work include the combined logic programming and DL knowledge base formalisms described beforehand and in the preliminaries of the upcoming chapter. The rest of this thesis can be roughly divided into two parts. The first part proposes a recently developed formalism called cq-programs for integrating rules with ontologies for the Semantic Web, which is an extension to dl-programs by more expressive rules, and is equipped with more expressive queries to the ontology layer. Moreover, it deals with our implementation of cq-programs, dubbed dl-plugin, as a software component for dlvhex, a reasoner for HEX-programs.

In dl-programs, the number of queries to the DL-reasoner is crucial for the efficiency of program execution. Since cq-programs allow for more expressive queries to ontologies, in particular conjunctive and union of conjunctive queries, before we start the evaluation of a program, we may bring together specific queries to form a fresh single query by rewriting the original program to a new, more efficient one. This has the effect that we have less queries to the ontology layer and thus process programs more efficiently. So, the second part of the thesis is concerned with optimization issues for cq-programs and provides experimental results.

This work is organized as follows:

- Chapter 2 provides the preliminaries for logic and answer-set programming, the family of description logics and its relationship to the OWL Web Ontology Language, the notion of dl-programs as an generalization of logic programming, HEX-programs as an extension of answer-set programming with higher order logic and external atoms, program unfolding as a means for optimizing logic programs, and conjunctive queries over description logics.
- In Chapter 3 we introduce the syntax and the semantics of a hybrid knowledge base formalism called cq-programs, which is composed of a logic program and a description logics part.
- The next chapter, 4, describes implementation details for our cq-program implementation dl-plugin for dlvhex, its connection to HEX-programs, and the usage patterns.
- Chapter 5 then provides the following contributions: equivalences for optimizing cq-programs, a generalization of partial evaluation of disjunctive programs presented in Section 2.7.2, the algorithms implementing the optimizing rewriting rules, and experimental results on program optimization in cq-programs.
- We give sample applications for cq-programs in Chapter 6.
- The conclusions in Chapter 7 summarizes this thesis and provides suggestions for further studies and future work.
- Proofs for the Theorems of Section 4 and Section 5 are given in Appendix A. Experimental data and the test setup of Section 5 is summarized in Appendix B.

Amanda:	And you, Sarek, would you also say thank you to your son?
Sarek:	I don't understand.
Amanda:	For saving your life.
Sarek:	Spock acted in the only logical manner open to him. One does not thank logic, Amanda.
Amanda:	Logic. Logic. I'm sick to death of logic. You know how I feel about your logic?
Sarek:	Emotional, isn't she? She's always been so.
Spock:	Indeed? Why did you marry her?
Sarek:	At the time, it seemed the logical thing to do.

—Star Trek, *Journey to Babel*, stardate 3842.4

2 Preliminaries

In this chapter, we will outline the basics and principles of answer-set programming and its existing implementations. Moreover, we will introduce description logics as a formal basis for ontology languages in the area of the Semantic Web, in particular the Web Ontology Language OWL. We then review dl-programs as the foundation of our new cq-program formalism. HEX-programs will serve as the formal basis for our implementation, hence they will be presented later on. In view of our optimization investigations on cq-programs, we provide the basis for unfolding logic programs and survey conjunctive queries over description logics. In the remainder of this thesis, we assume familiarity with first-order logic (cf. for instance [Boolos et al., 2003] or [van Dalen, 2004]).

2.1 Declarative Logic Programming

In computer science, programming languages can be classified into two big programming language concepts, one is the imperative and the other the declarative programming paradigm. Informally speaking, programming languages, which fall into the first paradigm, focus on how an algorithm solves a given problem. Such programming languages deal with states and sequences of operations, which can access and modify their state. Examples for imperative programming languages are C, C++, Java, and so forth.

On the other hand, a declarative programming language's centre of attention is the problem, hence such languages do not encode the sequence of operations. Instead, knowledge representing the problem is given as program and evaluated in an automated way. Usually, the programmer cannot determine how a solver for such a programming language process the program. Programming languages such as Prolog, Haskell, and Lisp, are considered declarative.

We focus hereby on the declarative style, concrete on *Declarative Logic Programming*. Here, the programmer identifies the relationships in a problem domain and states this in form of a logic program. In the following prototypical example, we clarify the difference between declarative and imperative programming style.

Example 2.1. The following procedure represents knowledge about birds:

```
function bird(x): boolean;
    if x = ’tweety’ then return true;
    else if x = ’sam’ then return true;
    else if penguin(x) then return true;
    else return false;
```

The corresponding logic program would be

$$P = \{bird(tweety), bird(sam), bird(X) \leftarrow penguin(X)\}$$

The `bird(x)` function is a typical imperative procedure. The associated knowledge is rather static—imagine we want to extend the knowledge by other types of birds or known bird individuals. This would amount to augment the if-then-else construct by various other conditions. On the other hand, the logic program can be easily extended just by adding the corresponding facts or rules.

But the true power of declarative logic programming is revealed when we want to reason about the encoded knowledge, for instance, when we want to know all bird instances. In the imperative procedure this is not possible, since the knowledge is hardcoded and cannot be retrieved, while we can easily add a query like “*bird(X)?*” to the logic program and retrieve all known birds.

As stated in the introduction, the widespread Prolog programs are not purely declarative. Program evaluation in Prolog depends on the order of the rules and the atoms in a rule body, which renders Prolog programs not always very comprehensible, especially when the programmer wants to modify the program. A different paradigm of logic programming that will be presented in the following is *answer-set programming* [Gelfond and Lifschitz, 1991], which allows to state purely declarative logic programs. Moreover, in answer-set programs, it is guaranteed that program evaluation always terminates, whereas Prolog programs do not have this characteristic.

2.2 Logic Programs under the Answer-Set Semantics

Answer-set programming stems from the stable model semantics of normal logic programs [Gelfond and Lifschitz, 1988] line of research (also known as *general logic programs*), which typically deals with negation as failure. This kind of negation is closely related to Reiter’s *Default Logic* [Reiter, 1980], hence it is also known as *default negation* or *weak negation.* Since negation as failure is different from *classical negation* (or *strong negation*) in classical logic, Gelfond and Lifschitz proposed a logic programming approach that allows for both negations [Gelfond and Lifschitz, 1990]. Subsequently, Gelfond and Lifschitz [1991] extended their semantics to disjunction in rule heads. Similar definitions for general logic programs and other classes of programs can be found in the literature (cf., e.g., [Lifschitz and Woo, 1992]). For an overview on other semantics for extended logic programs, see also [Dix, 1995].

Two prominent systems for computing answer sets are DLV [Eiter et al., 1998, 2000, Leone et al., 2006] and SMODELS [Niemelä, 1999, Simons et al., 2002], which allow for efficient declarative problem solving. The DLV system,[1] which is indirectly used in this

[1] http://www.dlvsystem.com/

thesis through the dlvhex framework,[2] has been developed for over a decade as joint work of the University of Calabria and Vienna University of Technology and is still actively maintained. For an in-detail discourse on `DLV` we refer to [Leone et al., 2006].

2.2.1 Syntax of answer-set programs

Let $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{X}$ be disjoint sets of predicate, constant, and variable symbols from a first-order vocabulary Φ, respectively, where $\mathcal{X}$ is infinite and $\mathcal{P}$ and $\mathcal{C}$ are countable. In accordance with common ASP solvers such as `DLV`, we assume that elements from $\mathcal{C}$ and $\mathcal{P}$ are string constants that begin with a lowercase letter or are double-quoted, where elements from $\mathcal{C}$ can also be integer numbers. Elements from $\mathcal{X}$ begin with an uppercase letter. A *term* is either a constant or a variable. Given $p \in \mathcal{P}$ an *atom* is defined as $p(t_1, \ldots, t_k)$, where k is called the arity of p and each $t_1, \ldots, t_k$ are terms. Atoms of arity $k = 0$ are called *propositional atoms.*

A *classical literal* (or simply *literal*) l is an atom p or a negated atom $\neg p$, where "$\neg$" is the symbol for true (classical) negation. Its *complementary* literal is $\neg p$ (resp., p). A *negation as failure literal* (or *NAF-literal*) is a literal l or a default-negated literal not l. Negation as failure is an extension to classical negation, denoting a fact as false if all attempts to prove it fail. Thus, *not* l evaluates to *true* if it cannot be foundedly demonstrated that l is true, i.e., if either l is false or we do not know whether l is true or false.

A *rule* r is an expression of the form

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n\,, \qquad k \geq 0\,, m \geq n \geq 0\,, \tag{2.1}$$

where $a_1, \ldots, a_k, b_1, \ldots, b_n$ are classical literals. We say that $a_1, \ldots, a_k$ is the *head* of r, while the conjunction $b_1, \ldots, b_m$, not $b_{m+1}, \ldots,$ not b_n is the *body* of r, where $b_1, \ldots, b_m$ (resp., not $b_{m+1}, \ldots,$ not b_n) is the *positive* (resp., *negative*) *body* of r. We use $H(r)$ to denote its head literals, and $B(r)$ to denote the set of all its body literals $B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$. A rule r without head literals (i.e., $k = 0$) is an *integrity constraint.* A rule r with exactly one head literal (i.e., $k = 1$) is a *normal rule.* If the body of r is empty (that is, $m = n = 0$), then r is a *fact*, and we often omit "$\leftarrow$".[3] An *extended disjunctive logic program* (EDLP, or simply *program*) P is a finite set of rules r of the form (2.1).

Programs without disjunction in the heads of rules are called *extended logic programs* (ELPs). A program P without negation as failure, i.e., for all $r \in P$, $B^-(r) = \emptyset$ is called *positive logic program.* If, additionally, no strong negation occurs in P, i.e., the only form of negation is default negation in rule bodies, then P is called a *normal logic program* (NLP). The generalization of an NLP by allowing default negation in the heads of rules is called *generalized logic program* (GLP). Additional program classes of logic programming with the corresponding restrictions on the rules in a program are summarized in Table 2.1. Program classes based on dependency information such as stratified programs [Apt et al., 1988] are not considered here.

2.2.2 Semantics of answer-set programs

The semantics of extended disjunctive logic programs is defined for variable-free programs. Thus, we first define the *ground instantiation* of a program that eliminates its variables.

The *Herbrand universe* of a program P, denoted HU_P, is the set of all constant symbols $C \subset \mathcal{C}$ appearing in P. If there is no such constant symbol, then $HU_P = \{c\}$, where c is

[2] `http://www.kr.tuwien.ac.at/research/dlvhex/`

[3] In this thesis, we will use both forms "$a \leftarrow$" and "$a.$" to denote that a is a fact in a logic program.

Name	restriction
definite horn	$k = 1$, $n = m$
horn	$k \leq 1$, $n = m$
normal	$k \leq 1$
definite	$k \geq 1$, $n = m$
positive	$n = m$
disjunctive	no restriction

Table 2.1: Program classes

an arbitrary constant symbol from Φ. As usual, terms, atoms, literals, rules, programs, etc. are *ground* iff they do not contain any variables. The *Herbrand base* of a program P, denoted HB_P, is the set of all ground (classical) literals that can be constructed from the predicate symbols appearing in P and the constant symbols in HU_P. A *ground instance* of a rule $r \in P$ is obtained from r by replacing every variable that occurs in r by a constant symbol from HU_P. We use $ground(P)$ to denote the set of all ground instances of rules in P.

The semantics for EDLPs is defined first for positive ground programs. A set of literals $X \subseteq HB_P$ is *consistent* iff $\{p, \neg p\} \not\subseteq X$ for every atom $p \in HB_P$. An *interpretation* I relative to a program P is a consistent subset of HB_P. We say that a set of literals S *satisfies* a rule r if $H(r) \cap S \neq \emptyset$ whenever $B^+(r) \subseteq S$ and $B^-(r) \cap S = \emptyset$. A *model* of a positive program P is an interpretation $I \subseteq HB_P$ such that I satisfies all rules in P. An *answer set* of a positive program P is the least model of P w.r.t. set inclusion.

To extend this definition to programs with negation as failure, we define the *Gelfond-Lifschitz transform* (also often called the *Gelfond-Lifschitz reduct*) of a program P relative to an interpretation $I \subseteq HB_P$, denoted P^I, as the ground positive program that is obtained from $ground(P)$ by

(i) deleting every rule r such that $B^-(r) \cap I \neq \emptyset$, and

(ii) deleting the negative body from every remaining rule.

An *answer set* of a program P is an interpretation $I \subseteq HB_P$ such that I is an answer set of P^I.

Example 2.2. Consider the following program P:

$$p \leftarrow \text{not } q.$$
$$q \leftarrow \text{not } p.$$

Let $I_1 = \{p\}$; then, $P^{I_1} = \{p \leftarrow\}$ with the unique model $\{p\}$ and thus I_1 is an answer set of P. Likewise, P has an answer set $\{q\}$. However, the empty set $\emptyset$ is not an answer set of P, since the respective reduct would be $\{p \leftarrow; q \leftarrow\}$ with the model $\{p, q\}$.

A constraint is used to eliminate "unwanted" models from the result, since its head is implicitly assumed to be *false*. A model that satisfies the body of a constraint is hence discarded from the set of answer sets.

Example 2.3. Let P be the program

$$p(X) \vee \neg p(X) \leftarrow q(X), \text{not}\, r(X).$$
$$q(c_1).\ r(c_2).$$

The grounding of P is

$$\begin{aligned}
p(c_1) \vee \neg p(c_1) &\leftarrow q(c_1), \text{not}\, r(c_1).\\
p(c_2) \vee \neg p(c_2) &\leftarrow q(c_2), \text{not}\, r(c_2).\\
q(c_1).\ r(c_2).
\end{aligned}$$

This program has several models. For instance, $I_1 = \{q(c_1), r(c_1), r(c_2), p(c_1)\}$ is a model of P, since P^{I_1} is just

$$q(c_1).\ r(c_2).$$

However, I_1 is not a minimal model of P^{I_1}. Now take $I_2 = \{q(c_1), r(c_2), p(c_1)\}$. We obtain P^{I_2} as

$$\begin{aligned}
p(c_1) \vee \neg p(c_1) &\leftarrow q(c_1).\\
q(c_1).\ r(c_2).
\end{aligned}$$

Indeed, I_2 is a minimal model of P^{I_2}, hence it is an answer set of P.

Example 2.4. Consider the 3COL example from Section 1.1. The grounding of P is the program $\mathit{ground}(P)$:

$$\begin{aligned}
\mathit{col}(a, \mathit{red}) \vee \mathit{col}(a, \mathit{green}) \vee \mathit{col}(a, \mathit{blue}) &\leftarrow \mathit{node}(a).\\
\mathit{col}(b, \mathit{red}) \vee \mathit{col}(b, \mathit{green}) \vee \mathit{col}(b, \mathit{blue}) &\leftarrow \mathit{node}(b).\\
\mathit{col}(c, \mathit{red}) \vee \mathit{col}(c, \mathit{green}) \vee \mathit{col}(c, \mathit{blue}) &\leftarrow \mathit{node}(c).\\
\mathit{col}(d, \mathit{red}) \vee \mathit{col}(d, \mathit{green}) \vee \mathit{col}(d, \mathit{blue}) &\leftarrow \mathit{node}(d).\\
\mathit{col}(e, \mathit{red}) \vee \mathit{col}(e, \mathit{green}) \vee \mathit{col}(e, \mathit{blue}) &\leftarrow \mathit{node}(e).\\
\mathit{col}(\mathit{blue}, \mathit{red}) \vee \mathit{col}(\mathit{blue}, \mathit{green}) \vee \mathit{col}(\mathit{blue}, \mathit{blue}) &\leftarrow \mathit{node}(\mathit{blue}).\\
\mathit{col}(\mathit{green}, \mathit{red}) \vee \mathit{col}(\mathit{green}, \mathit{green}) \vee \mathit{col}(\mathit{green}, \mathit{blue}) &\leftarrow \mathit{node}(\mathit{green}).\\
\mathit{col}(\mathit{red}, \mathit{red}) \vee \mathit{col}(\mathit{red}, \mathit{green}) \vee \mathit{col}(\mathit{red}, \mathit{blue}) &\leftarrow \mathit{node}(\mathit{red}).\\
&\leftarrow \mathit{col}(a, a), \mathit{col}(a, a), \mathit{edge}(a, a).\\
&\leftarrow \mathit{col}(a, a), \mathit{col}(b, a), \mathit{edge}(a, b).\\
&\ \ \vdots\\
&\leftarrow \mathit{col}(\mathit{red}, a), \mathit{col}(b, a), \mathit{edge}(a, b).\\
&\ \ \vdots\\
\mathit{node}(a) &\leftarrow \mathit{edge}(a, a).\\
\mathit{node}(a) &\leftarrow \mathit{edge}(a, b).\\
&\ \ \vdots\\
\mathit{node}(b) &\leftarrow \mathit{edge}(a, b).\\
&\ \ \vdots\\
\mathit{edge}(a, b).\ \mathit{edge}(b, c).\ \mathit{edge}(c, d).\ &\mathit{edge}(d, a).\ \mathit{edge}(e, c).\ \mathit{edge}(e, b).
\end{aligned}$$

Since P is positive, for each Herbrand interpretation I, $P^I = \mathit{ground}(P)$. Hence, the minimal models of $\mathit{ground}(P)$ and the answer sets of P coincide.

The main reasoning tasks that are associated with EDLPs under the answer-set semantics are the following:

- decide whether a given program P has an answer set;
- given a program P and a ground formula ϕ, decide whether ϕ holds in every (resp., some) answer set of P (*cautious* (resp., *brave*) *reasoning*);
- given a program P and an interpretation $I \subseteq HB_P$, decide whether I is an answer set of P (*answer-set checking*); and
- compute the set of all answer sets of a given program P.

2.3 Description Logics

Description logics arise from scientific research on two prominent knowledge representation languages. In the 1970s, *Semantic Networks* were a widely used formalism for representing relationships between concepts, especially in natural language processing. *Frame-based systems* appeared in the beginning of the 1980s, which is a similar formalism. Both languages lack a formal semantics, hence they are non-logical formalisms with limited reasoning ability. In fact, reasoning in such languages was done by manipulating ad hoc data structures.

Later on, a semantic characterisation based on first-order logic was given to frame systems, which subsequently led to *terminological systems* and *concept languages*. The term *Description Logics* was coined to emphasize the logical underpinning of this line of research. There are multiple variants of this kind of formalism, so called families of description logics; the plural form "logics" stems from this observation.

Nowadays, description logics are heavily used in ontology and Semantic Web research (see also Section 1.2). For excellent introductions to description logics, we refer to [Baader et al., 2003, Baader and Lutz, 2006, Baader et al., 2007], which, too, deal with the close relationship of description logics and modal logics.

Applications for DLs are manifold. Beside the obvious connection to ontologies for the Semantic Web, DLs are useful for reasoning with ER data models [Chen, 1976], the standard approach for conceptual modelling in database design. By carefully translating an ER schema into a DL-KB [Baader et al., 2003], one can check the consistency of an ER schema by reasoning methods employed by DL-reasoners. Another application is reasoning with UML [Berardi et al., 2005], the standard language for modelling software. Additionally, description logics are big players in the area of bioinformatics, where lots of ontologies have been developed. Three prominent ontologies for describing biomedical terminologies are SNOMED,[4] GALEN,[5] and NCI.[6]

In this section, we reminisce back on the foundations of the two description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, which are the underpinning of Semantic Web ontology languages and provide the basis for our novel types of answer-set programs that will be presented in Section 2.5 and Chapter 3.

The naming scheme for particular description logics are in match with the logical primitives they provide. Basic boolean operators like concept union $\sqcup$ and concept intersection $\sqcap$, modality operators like $\exists R.C$ and $\forall R.C$, and negation $\neg$ are considered present (the corresponding DL is called $\mathcal{ALC}$). $\mathcal{SHOIN}(\mathbf{D})$ now provides additional operators for role transitivity ($\mathcal{S}$), role hierarchy ($\mathcal{H}$), nominals or "one-of"-constructor ($\mathcal{O}$), role inverses ($\mathcal{I}$), unqualified number restrictions ($\mathcal{N}$), and datatypes ($\mathbf{D}$).

[4]`http://www.snomed.org/`
[5]`http://www.opengalen.org/`
[6]`http://www.cancer.gov/`

The $\mathcal{SHIF}(\mathbf{D})$ DL is less expressive, $\mathcal{F}$ stands for functionality, which is a restricted form of number restriction $\leq 1R$. $\mathcal{F}$ is subsumed by $\mathcal{N}$ of $\mathcal{SHOIN}(\mathbf{D})$. Consequently, $\mathcal{SHIF}(\mathbf{D})$ is a restriction to the $\mathcal{SHOIN}(\mathbf{D})$ language, which have a close connection to the description logics $\mathcal{SHOQ}(\mathbf{D})$ [Horrocks and Sattler, 2001] and $\mathcal{SHOIQ}(\mathbf{D})$ [Horrocks and Sattler, 2005].

Syntax and semantics definitions for description logics can be given in various ways, for instance by a translation into first-order logic. Concrete, the DL $\mathcal{ALC}$ can be mapped into the first-order logic fragment $\mathcal{L}^2$, i.e., into first-order logic over unary and binary predicates with at most two variables. For further material and details, we refer to [Baader et al., 2003]. An in-depth introduction to $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ and their relationship to the OWL language is given in [Horrocks and Patel-Schneider, 2003, Horrocks et al., 2003]. For our purposes, we use the definition of [Eiter et al., 2007b].

2.3.1 Syntax of $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$

We now recall the syntax of $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$. We first describe the syntax of the latter, which has the following datatypes and elementary ingredients. We assume a set of *elementary datatypes* and a set of *data values*. A *datatype* is either an elementary datatype or a set of data values (called *datatype oneOf*). A *datatype theory* $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$ consists of a *datatype* (or *concrete*) *domain* $\Delta^{\mathbf{D}}$ and a mapping $\cdot^{\mathbf{D}}$ that assigns to every elementary datatype a subset of $\Delta^{\mathbf{D}}$ and to every data value an element of $\Delta^{\mathbf{D}}$. The mapping $\cdot^{\mathbf{D}}$ is extended to all datatypes by $\{v_1, \ldots\}^{\mathbf{D}} = \{v_1^{\mathbf{D}}, \ldots\}$. Let $\mathbf{A}$, $\mathbf{R}_A$, $\mathbf{R}_D$, and $\mathbf{I}$ be pairwise disjoint finite nonempty sets of *atomic concepts*, *abstract roles*, *datatype* (or *concrete*) *roles*, and *individuals*, respectively. We denote by $\mathbf{R}_A^-$ the set of inverses R^- of all $R \in \mathbf{R}_A$.

Roles and concepts are defined as follows. A *role* is an element of $\mathbf{R}_A \cup \mathbf{R}_A^- \cup \mathbf{R}_D$. *Concepts* are inductively defined as follows. Every atomic concept $C \in \mathbf{A}$ is a concept. If $o_1, o_2, \ldots$ are individuals from $\mathbf{I}$, then $\{o_1, o_2, \ldots\}$ is a concept (called *oneOf*). If C and D are concepts, then also $(C \sqcap D)$, $(C \sqcup D)$, and $\neg C$ are concepts (called *conjunction*, *disjunction*, and *negation*, respectively). If C is a concept, R is an abstract role from $\mathbf{R}_A \cup \mathbf{R}_A^-$, and n is a nonnegative integer, then $\exists R.C$, $\forall R.C$, $\geq nR$, and $\leq nR$ are concepts (called *exists*, *value*, *atleast*, and *atmost restriction*, respectively). If D is a datatype, U is a datatype role from $\mathbf{R}_D$, and n is a nonnegative integer, then $\exists U.D$, $\forall U.D$, $\geq nU$, and $\leq nU$ are concepts (called *datatype exists*, *value*, *atleast*, and *atmost restriction*, respectively). We use $\top$ and $\bot$ to abbreviate the concepts $C \sqcup \neg C$ and $C \sqcap \neg C$, respectively, and we eliminate parentheses as usual.

We next define axioms and knowledge bases as follows. An *axiom* is an expression of one of the following forms: (1) $C \sqsubseteq D$ (called *concept inclusion axiom*), where C and D are concepts; (2) $R \sqsubseteq S$ (called *role inclusion axiom*), where either $R, S \in \mathbf{R}_A$ or $R, S \in \mathbf{R}_D$; (3) $\mathrm{Trans}(R)$ (called *transitivity axiom*), where $R \in \mathbf{R}_A$; (4) $C(a)$ (called *concept membership axiom*), where C is a concept and $a \in \mathbf{I}$; (5) $R(a, b)$ (resp., $U(a, v)$) (called *role membership axiom*), where $R \in \mathbf{R}_A$ (resp., $U \in \mathbf{R}_D$) and $a, b \in \mathbf{I}$ (resp., $a \in \mathbf{I}$ and v is a data value); and (6) $a = b$ (resp., $a \neq b$) (called *equality* (resp., *inequality*) *axiom*), where $a, b \in \mathbf{I}$. A (*description logic*) *knowledge base* L is a finite set of axioms.

For an abstract role $R \in \mathbf{R}_A$, we define $\mathrm{Inv}(R) = R^-$ and $\mathrm{Inv}(R^-) = R$. Let the transitive and reflexive closure of $\sqsubseteq$ on abstract roles relative to L, denoted $\sqsubseteq^\star$, be defined as follows. For two abstract roles R and S in L, let $S \sqsubseteq^\star R$ relative to L iff either (a) $S = R$, (b) $S \sqsubseteq R \in L$, (c) $\mathrm{Inv}(S) \sqsubseteq \mathrm{Inv}(R) \in L$, or (d) some abstract role Q exists such that $S \sqsubseteq^\star Q$ and $Q \sqsubseteq^\star R$ relative to L. An abstract role R is *simple* relative to L iff for each abstract role S such that $S \sqsubseteq^\star R$ relative to L, it holds that (i) $\mathrm{Trans}(S) \notin L$ and

(ii) Trans(Inv(S)) $\notin L$. For decidability, number restrictions in L are restricted to simple abstract roles [Horrocks et al., 1999].

Observe that in $\mathcal{SHOIN}(\mathbf{D})$, concept and role membership axioms can equally be expressed through concept inclusion axioms. The knowledge that the individual a is an instance of the concept C can be expressed by the concept inclusion axiom $\{a\} \sqsubseteq C$, while the knowledge that the pair (a, b) (resp., (a, v)) is an instance of the role R (resp., U) can be expressed by $\{a\} \sqsubseteq \exists R.\{b\}$ (resp., $\{a\} \sqsubseteq \exists U.\{v\}$).

The syntax of $\mathcal{SHIF}(\mathbf{D})$ is as the above syntax of $\mathcal{SHOIN}(\mathbf{D})$, but without the oneOf constructor and with the atleast and atmost constructors limited to 0 and 1.

Note that other definitions of description logic knowledge bases exist. A widely used definition is the notion of DL-KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, where $\mathcal{T}$ is called *TBox* and consists of a set of concept inclusion axioms (the terminological knowledge), $\mathcal{R}$ is called *RBox* and consists of a set of axiom of form (2) or (3) (the role hierarchy), and $\mathcal{A}$ is called *ABox* and consists of a set of concept or role membership axioms (the assertional knowledge, or extensional part). $\mathcal{T}$ and $\mathcal{R}$ builds the intensional part of a DL-KB. We do not use this clear separation in our framework, but sometimes refer to the extensional part of L as ABox and the intentional part of L as TBox.

2.3.2 Semantics of $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$

We now define the semantics of $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ in terms of general first-order interpretations, as usual, and we recall some important reasoning problems in description logics.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with respect to a datatype theory $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$ consists of a nonempty (*abstract*) *domain* $\Delta^{\mathcal{I}}$ disjoint from $\Delta^{\mathbf{D}}$, and a mapping $\cdot^{\mathcal{I}}$ that assigns to each atomic concept $C \in \mathbf{A}$ a subset of $\Delta^{\mathcal{I}}$, to each individual $o \in \mathbf{I}$ an element of $\Delta^{\mathcal{I}}$, to each abstract role $R \in \mathbf{R}_A$ a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to each datatype role $U \in \mathbf{R}_D$ a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathbf{D}}$. The mapping $\cdot^{\mathcal{I}}$ is extended to all concepts and roles as usual (where $\#S$ denotes the cardinality of a set S):

- $(R^-)^{\mathcal{I}} = \{(a, b) \mid (b, a) \in R^{\mathcal{I}}\}$;
- $\{o_1, \ldots, o_n\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \ldots, o_n^{\mathcal{I}}\}$;
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, and $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$;
- $(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y\colon (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$;
- $(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y\colon (x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$;
- $({\geq}nR)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x, y) \in R^{\mathcal{I}}\}) \geq n\}$;
- $({\leq}nR)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x, y) \in R^{\mathcal{I}}\}) \leq n\}$;
- $(\exists U.D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y\colon (x, y) \in U^{\mathcal{I}} \wedge y \in D^{\mathbf{D}}\}$;
- $(\forall U.D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y\colon (x, y) \in U^{\mathcal{I}} \rightarrow y \in D^{\mathbf{D}}\}$;
- $({\geq}nU)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x, y) \in U^{\mathcal{I}}\}) \geq n\}$;
- $({\leq}nU)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x, y) \in U^{\mathcal{I}}\}) \leq n\}$.

The *satisfaction* of a description logic axiom F in the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with respect to $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$, denoted $\mathcal{I} \models F$, is defined as follows:

- $\mathcal{I} \models C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$;
- $\mathcal{I} \models R \sqsubseteq S$ iff $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$;
- $\mathcal{I} \models \text{Trans}(R)$ iff $R^{\mathcal{I}}$ is transitive;
- $\mathcal{I} \models C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$;
- $\mathcal{I} \models R(a,b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ (resp., $\mathcal{I} \models U(a,v)$ iff $(a^{\mathcal{I}}, v^{\mathbf{D}}) \in U^{\mathcal{I}}$); and
- $\mathcal{I} \models a = b$ iff $a^{\mathcal{I}} = b^{\mathcal{I}}$ (resp., $\mathcal{I} \models a \neq b$ iff $a^{\mathcal{I}} \neq b^{\mathcal{I}}$).

The interpretation $\mathcal{I}$ *satisfies* the axiom F, or $\mathcal{I}$ is a *model* of F, iff $\mathcal{I} \models F$. The interpretation $\mathcal{I}$ *satisfies* a knowledge base L, or $\mathcal{I}$ is a *model* of L, denoted $\mathcal{I} \models L$, iff $\mathcal{I} \models F$ for all $F \in L$. We say that L is *satisfiable* (resp., *unsatisfiable*) iff L has a (resp., no) model. An axiom F is a *logical consequence* of L, denoted $L \models F$, iff every model of L satisfies F. A negated axiom $\neg F$ is a *logical consequence* of L, denoted $L \models \neg F$, iff every model of L does not satisfy F.

Some important reasoning problems related to description logic knowledge bases L are the following:

(1) decide whether a given L is satisfiable;

(2) given L and a concept C, decide whether $L \not\models C \sqsubseteq \bot$;

(3) given L and two concepts C and D, decide whether $L \models C \sqsubseteq D$;

(4) given L, an individual $a \in \mathbf{I}$, and a concept C, decide whether $L \models C(a)$; and

(5) given L, two individuals $a, b \in \mathbf{I}$ (resp., an individual $a \in \mathbf{I}$ and a data value v), and an abstract role $R \in \mathbf{R}_A$ (resp., a datatype role $U \in \mathbf{R}_D$), decide whether $L \models R(a,b)$ (resp., $L \models U(a,v)$).

Here, (1) is a special case of (2), since L is satisfiable iff $L \not\models \top \sqsubseteq \bot$. Furthermore, (2) and (3) can be reduced to each other, since $L \models C \sqcap \neg D \sqsubseteq \bot$ iff $L \models C \sqsubseteq D$. Finally, in $\mathcal{SHOIN}(\mathbf{D})$, (4) and (5) are special cases of (3). In Section 2.8, we present another important reasoning problem, viz., conjunctive queries over description logics. This problem will be used for our cq-program formalism (see Section 3).

Example 2.5. Consider the following description logic knowledge base L with knowledge about space tourists:

$$\geq 1 \mathit{motherOf} \sqsubseteq \mathit{Female} \tag{1}$$
$$\geq 1 \mathit{fatherOf} \sqsubseteq \mathit{Male} \tag{2}$$
$$\mathit{Male} \sqsubseteq \neg \mathit{Female} \sqcup \mathit{SpaceTourist} \tag{3}$$
$$\mathit{SpaceTourist} \sqsubseteq \mathit{Female} \sqcup \mathit{Male} \tag{4}$$
$$\mathit{Female} \sqcap \mathit{SpaceTourist}(\mathit{Anousheh}) \tag{5}$$
$$\mathit{Male} \sqcap \mathit{SpaceTourist}(\mathit{Mark}) \tag{6}$$
$$\mathit{fatherOf}(\mathit{Rick}, \mathit{Mark}) \tag{7}$$
$$\mathit{motherOf}(\mathit{Fakhri}, \mathit{Anousheh}) \tag{8}$$

The general concept inclusions (1)–(4) form the TBox of L; the assertions (5)–(8) are the ABox of L.

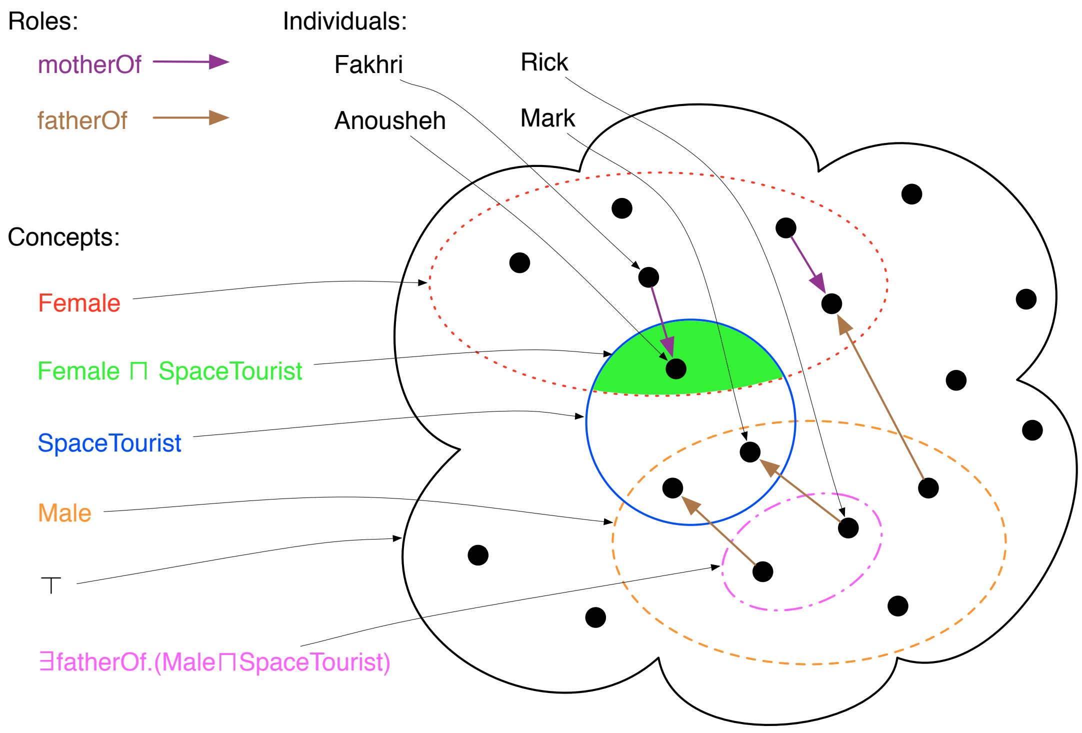

Figure 2.1: Space tourists knowledge base

Axiom (1) and (2) set the domain of *motherOf* and *fatherOf* to the concept *Female* and *Male*, respectively. Hence, when we assert that someone is a mother of a person, we implicitly assure that this mother is known to be a female person. In (3), we define that male persons are not female or they are spacetourists. The intuition behind (4) is that spacetourists could be female or male. Assertions (5) declares individual Anousheh as female spacetourist, while (6) states that Mark is a male spacetourist. In (7) and (8), we define parenthood relationships; Rick is the father of Mark and Fakhri is the mother of Anousheh by (7) and (8), resp.

Now we can reason about this knowledge of space tourists. For instance, we can infer that Fakhri is female, i.e., $L \models \mathit{Female}(\mathit{Fakhri})$ by (1) and (8). Similarly, $L \models \exists \mathit{fatherOf}.(\mathit{Male} \sqcap \mathit{SpaceTourist})(\mathit{Rick})$.

A graphical representation for a particular model of L is shown in Figure 2.1. As can be seen, there is much more information present than represented in the DL-KB. For instance, a female individual, lets call it f, has a known mother and a known father. Note that $L \not\models \mathit{Female}(f)$, because in another model of L, f might not be present in $\Delta^{\mathcal{I}}$, hence not every model $\mathcal{I}$ satisfies $\mathit{Female}(f)$.

2.4 Web Ontology Language

Ontologies play a significant role in the Semantic Web. They define the concepts and relationships in a domain of interest to build vocabularies ready for comprehensive deployment. In [Gruber, 1993], formal ontologies were identified as entities for specifying sharable and reusable knowledge. Due to the far reaching design goals of the Semantic Web, such reusable ontologies act as a key technology for the feasibility of the Semantic Web vision. Additionally, ontologies are more and more understood as convenient tools for specifying knowledge in interdisciplinary sciences. The applications presented in [Horrocks, 2005] and in Section 6, too, record this movement, hence the Semantic Web with its technological advancements support these emerging tasks.

The *Web Ontology Language* (OWL) [Bechhofer et al., 2004] has been developed by the W3C Web Ontology Working Group.[7] The OWL specification appeared as W3C Recommendation in February 2004. Several ancestors of OWL can be identified: *Simple HTML Ontology Extensions* (SHOE) [Heflin and Hendler, 2000], *DARPA Agent Modelling Language* (DAML) [Hendler and McGuiness, 2000], *Ontology Inference Layer* (OIL) [Fensel et al., 2001], and DAML+OIL [Horrocks, 2002b,a]. SHOE was an extension of HTML with semantic markup to represent ontologies in hypertext documents. Later, the DAML and OIL languages were combined to form DAML+OIL, an RDF-S based ontology language. These efforts eventually resulted in the OWL family of ontology languages, now the standard language for specifying ontologies on the Semantic Web. One design goal for OWL was to maintain as much compatibility to its preceding formalisms as possible. See also [Horrocks et al., 2003] for more information on the relationship between OWL and other ontology formalisms.

OWL consists of three sublanguages with increasing expressivity: *OWL Lite*, *OWL DL*, and *OWL Full*. OWL Lite and OWL DL semantics are based on description logics, namely, $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, respectively. OWL Full on the other hand loosens specific syntactic restrictions of OWL Lite and OWL DL, hence reasoning in OWL Full is undecidable. An overview on the differences between the various sublanguages is given in `http://www.w3.org/TR/owl-ref/#Sublanguage-def`. Syntax and semantics of OWL is presented in [Patel-Schneider et al., 2004] and summarized in the tables below, which are from [Horrocks et al., 2003].

Syntax and semantics of OWL

The abstract syntax for class descriptions and axioms in OWL DL ontologies is given in the first column of Table 2.2 and Table 2.3, respectively. The syntax for OWL Lite is basically the same, with the following restrictions:

- `oneOf`, `unionOf`, and `complementOf` are prohibited;
- `maxCardinality`, `minCardinality`, and `cardinality` restrictions may only have 0 and 1 as parameter n;
- `hasValue` restrictions are prohibited; and
- `EnumeratedClass` and `DisjointClasses` axioms are not allowed.

The concrete constraints on the syntax of OWL Lite ontologies are summarized in `http://www.w3.org/TR/owl-ref/#OWLLite`.

Note that we left out `AnnotationProperty` and `OntologyProperty` axioms, since they do not have an immediate DL equivalent expression. In fact, those property axioms are merely used for annotating extralogical information in the ontology like authorship information and textual descriptions for class and property axioms. As an aside, the `owl:imports` built-in ontology property can be used to include other ontologies (see also the discussion in [Horrocks et al., 2003]).

The second column of Table 2.2 and 2.3 maps OWL abstract syntax to the corresponding DL syntax, hence OWL ontologies can be imagined as syntactic variants of description logics.

[7]`http://www.w3.org/2001/sw/WebOnt/`

Abstract Syntax	DL Syntax	Semantics
Descriptions (C)		
A (URI reference)	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
`owl:Thing`	$\top$	$\texttt{owl:Thing}^{\mathcal{I}} = \Delta^{\mathcal{I}}$
`owl:Nothing`	$\bot$	$\texttt{owl:Nothing}^{\mathcal{I}} = \emptyset$
`intersectionOf`(C_1 C_2 ...)	$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
`unionOf`(C_1 C_2 ...)	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
`complementOf`(C)	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
`oneOf`(o_1 ...)	$\{o_1, \ldots\}$	$\{o_1, \ldots\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \ldots\}$
`restriction(`R `someValuesFrom(`C`))`	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
`restriction(`R `allValuesFrom(`C`))`	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
`restriction(`R `hasValue(`o`))`	$R : o$	$(\forall R.o)^{\mathcal{I}} = \{x \mid \langle x, o^{\mathcal{I}}\rangle \in R^{\mathcal{I}}\}$
`restriction(`R `minCardinality(`n`))`	$\geq nR$	$(\geq nR)^{\mathcal{I}} = \{x \mid \sharp(\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\}) \geq n\}$
`restriction(`R `maxCardinality(`n`))`	$\leq nR$	$(\leq nR)^{\mathcal{I}} = \{x \mid \sharp(\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\}) \leq n\}$
`restriction(`R `cardinality(`n`))`	$= nR$	$(= nR)^{\mathcal{I}} = \{x \mid \sharp(\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\}) = n\}$
`restriction(`U `someValuesFrom(`D`))`	$\exists U.D$	$(\exists U.D)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in U^{\mathcal{I}} \wedge y \in D^{\mathbf{D}}\}$
`restriction(`U `allValuesFrom(`D`))`	$\forall U.D$	$(\forall U.D)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in U^{\mathcal{I}} \rightarrow y \in D^{\mathbf{D}}\}$
`restriction(`U `hasValue(`v`))`	$U : v$	$(U : v)^{\mathcal{I}} = \{x \mid \langle x, v^{\mathcal{I}}\rangle \in U^{\mathcal{I}}\}$
`restriction(`U `minCardinality(`n`))`	$\geq nU$	$(\geq nU)^{\mathcal{I}} = \{x \mid \sharp(\{y \mid \langle x, y\rangle \in U^{\mathcal{I}}\}) \geq n\}$
`restriction(`U `maxCardinality(`n`))`	$\leq nU$	$(\leq nU)^{\mathcal{I}} = \{x \mid \sharp(\{y \mid \langle x, y\rangle \in U^{\mathcal{I}}\}) \leq n\}$
`restriction(`U `cardinality(`n`))`	$= nU$	$(= nU)^{\mathcal{I}} = \{x \mid \sharp(\{y \mid \langle x, y\rangle \in U^{\mathcal{I}}\}) = n\}$
Data Ranges (D)		
D (URI reference)	D	$D^{\mathbf{D}} \subseteq \Delta^{\mathbf{D}}$
`oneOf`($v_1 \ldots$)	$\{v_1, \ldots\}$	$\{v_1, \ldots\}^{\mathcal{I}} = \{v_1^{\mathcal{I}}, \ldots\}$
Object Properties (R)		
R (URI reference)	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	R^-	$(R^-)^{\mathcal{I}} = (R^{\mathcal{I}})^-$
Datatype Properties (U)		
U (URI reference)	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathbf{D}}$
Individuals (o)		
o (URI reference)	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Data Values (v)		
v (RDF literal)	v	$v^{\mathcal{I}} = v^{\mathbf{D}}$

Table 2.2: OWL DL Syntax vs. DL Syntax and Semantics

Abstract Syntax	DL Syntax	Semantics
`Class(`A `partial` C_1 ... C_n`)`	$A \sqsubseteq C_1 \sqcap \cdots \sqcap C_n$	$A^\mathcal{I} \subseteq C_1^\mathcal{I} \cap \cdots \cap C_n^\mathcal{I}$
`Class(`A `complete` C_1 ... C_n`)`	$A = C_1 \sqcap \cdots \sqcap C_n$	$A^\mathcal{I} = C_1^\mathcal{I} \cap \cdots \cap C_n^\mathcal{I}$
`EnumeratedClass(`A o_1 ... o_n`)`	$A = \{o_1, \ldots, o_n\}$	$A^\mathcal{I} = \{o_1^\mathcal{I}, \ldots, o_n^\mathcal{I}\}$
`SubClassOf(`C_1 C_2`)`	$C_1 \sqsubseteq C_2$	$C_1^\mathcal{I} \subseteq C_2^\mathcal{I}$
`EquivalentClasses(`C_1 ... C_n`)`	$C_1 = \cdots = C_n$	$C_1^\mathcal{I} = \cdots = C_n^\mathcal{I}$
`DisjointClasses(`C_1 ... C_n`)`	$C_i \sqcap C_j = \bot, i \neq j$	$C_i^\mathcal{I} \cap C_j^\mathcal{I} = \emptyset, i \neq j$
`Datatype(`D`)`		$D^\mathcal{I} \subseteq \Delta^\mathbf{D}$
`DatatypeProperty(` U		
`super(`U_1`)...super(`U_n`)`	$U \sqsubseteq U_i$	$U^\mathcal{I} \subseteq U_i^\mathcal{I}$
`domain(`C_1`)...domain(`C_m`)`	$\geq 1\, U \sqsubseteq C_i$	$U^\mathcal{I} \subseteq C_i^\mathcal{I} \times \Delta^\mathbf{D}$
`range(`D_1`)...range(`D_l`)`	$\top \sqsubseteq \forall U.D_i$	$U^\mathcal{I} \subseteq \Delta^\mathcal{I} \times D_i^\mathcal{I}$
`[Functional])`	$\top \sqsubseteq \leq 1\, U$	$U^\mathcal{I}$ is functional
`SubPropertyOf(`U_1 U_2`)`	$U_1 \sqsubseteq U_2$	$U_1^\mathcal{I} \subseteq U_2^\mathcal{I}$
`EquivalentProperties(`U_1 ... U_n`)`	$U_1 = \cdots = U_n$	$U_1^\mathcal{I} = \cdots = U_n^\mathcal{I}$
`ObjectProperty(` R		
`super(`R_1`)...super(`R_n`)`	$R \sqsubseteq R_i$	$R^\mathcal{I} \subseteq R_i^\mathcal{I}$
`domain(`C_1`)...domain(`C_m`)`	$\geq 1\, R \sqsubseteq C_i$	$R^\mathcal{I} \subseteq C_i^\mathcal{I} \times \Delta^\mathcal{I}$
`range(`C_1`)...range(`C_l`)`	$\top \sqsubseteq \forall R.C_i$	$R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times C_i^\mathcal{I}$
`[inverseOf(`R_0`)]`	$R = R_0^-$	$R^\mathcal{I} = (R_0^\mathcal{I})^-$
`[Symmetric]`	$R = R^-$	$R^\mathcal{I} = (R^\mathcal{I})^-$
`[Functional]`	$\top \sqsubseteq \leq 1\, R$	$R^\mathcal{I}$ is functional
`[InverseFunctional]`	$\top \sqsubseteq \leq 1\, R^-$	$(R^\mathcal{I})^-$ is functional
`[Transitive])`	$Tr(R)$	$R^\mathcal{I} = (R^\mathcal{I})^+$
`SubPropertyOf(`R_1 R_2`)`	$R_1 \sqsubseteq R_2$	$R_1^\mathcal{I} \subseteq R_2^\mathcal{I}$
`EquivalentProperties(`R_1 ... R_n`)`	$R_1 = \ldots = R_n$	$R_1^\mathcal{I} = \ldots = R_n^\mathcal{I}$
`Individual(`o `type(`C_1`)...type(`C_n`)`	$o \in C_i$	$o^\mathcal{I} \in C_i^\mathcal{I}$
`value(`R_1 o_1`)...value(`R_n o_n`)`	$\langle o, o_i \rangle \in R_i$	$\langle o^\mathcal{I}, o_i^\mathcal{I} \rangle \in R_i^\mathcal{I}$
`value(`U_1 v_1`)...value(`U_n v_n`))`	$\langle o, v_i \rangle \in R_i$	$\langle o^\mathcal{I}, v_i^\mathcal{I} \rangle \in R_i^\mathcal{I}$
`SameIndividual(`o_1 ... o_n`)`	$o_1 = \cdots = o_n$	$o_1^\mathcal{I} = \cdots = o_n^\mathcal{I}$
`DifferentIndividuals(`o_1 ... o_n`)`	$o_i \neq o_j, i \neq j$	$o_i^\mathcal{I} \neq o_j^\mathcal{I}, i \neq j$

Table 2.3: OWL DL Axioms and Facts

Moreover, OWL ontologies are in general just RDF graphs, consequently they may come in form of RDF/XML,[8] the usual way to denote OWL ontologies. The next example will show that this particular manner for denoting OWL ontologies is not designed for human-readability, instead, RDF/XML is built for an easy machine-to-machine communication.

Example 2.6. To give an example for an OWL ontology, we translate the DL knowledge base L in Example 2.5 into OWL abstract syntax. The OWL language constructors in use fit nicely into the OWL DL language.

```
SubClassOf(restriction(motherOf minCardinality(1)) Female)
SubClassOf(restriction(fatherOf minCardinality(1)) Male)
SubClassOf(Male unionOf(complementOf(Female) SpaceTourist))
SubClassOf(SpaceTourist unionOf(Female Male))
Individual(Anousheh type(intersectionOf(Female SpaceTourist)))
Individual(Mark type(intersectionOf(Male SpaceTourist)))
Individual(Rick value(fatherOf Mark))
Individual(Fakhri value(motherOf Anousheh))
```

The concrete syntax of OWL is much more verbose, for instance, axiom (3) in Example 2.5 has the following RDF/XML serialization:

```
<owl:Class rdf:about="#Male">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class>
          <owl:complementOf rdf:resource="#Female"/>
        </owl:Class>
        <owl:Class rdf:about="#SpaceTourist"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

The semantics for OWL DL and OWL Lite is presented in the third column of Table 2.2 and Table 2.3. As remarked in the beginning of this section, OWL Lite and OWL DL ontologies correspond to $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ DL-KBs, respectively. OWL uses the RDF datatyping scheme to refer to datatypes.[9] Hence, OWL uses XML Schema datatypes like `xsd:string` or `xsd:float`. In a model $\mathcal{I}$, $\Delta^{\mathcal{I}}$ is the domain of individuals and $\Delta^{\mathbf{D}}$ is the domain of data values (cf. Section 2.3).

2.5 dl-Programs

This section provides the preliminaries for dl-programs, which have been first introduced in [Eiter et al., 2004b]. Such programs build the theoretical framework for a novel type of combined description logic and logic programming knowledge bases. Hence, they form another contribution to the attempt in finding an appropriate formalisms for combined rules and ontologies for the Semantic Web. In this thesis, we will use dl-programs as a

[8]http://www.w3.org/TR/rdf-syntax-grammar/
[9]http://www.w3.org/TR/rdf-concepts/

starting point for our combined knowledge base formalism. Several impediments occur when building such combined knowledge bases, see Section 1.3 for an overview.

Description Logic programs (dl-programs) consists of a normal logic program P and a DL-KB L. The logic program P might contain special devices called dl-atoms. Those dl-atoms may occur in the body of a rule and involve queries to a DL-KB. Moreover, dl-atoms can specify an input to L before querying the external DL-KB, thus dl-programs allow for an bidirectional data flow between the description logic component and the logic program.

The way dl-programs interface DL-KBs allow them to act as loosely coupled formalism. This feature brings the advantage of reusing existing logic programming and DL system in order to build an implementation of dl-programs.

In the following, we provide the syntax of dl-programs and an overview of the semantics. An in-detail treatise is given in [Eiter et al., 2007b].

Syntax and semantics of dl-programs

Informally, a dl-program consists of a description logic knowledge base L and a generalized normal program P, which may contain queries to L. Roughly, such a query asks whether a specific description logic axiom is entailed by L or not.

We first define dl-queries and dl-atoms, which are used to express queries to the description logic knowledge base L. A *dl-query*[10] $Q(\mathbf{t})$ is either

- a concept inclusion axiom F or its negation $\neg F$, or
- of the forms $C(t)$ or $\neg C(t)$, where C is a concept and t is a term, or
- of the forms $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, where R is a role and t_1, t_2 are terms.

A *dl-atom* has the form

$$\mathrm{DL}[S_1\, op_1 p_1, \ldots, S_m\, op_m\, p_m; Q](\mathbf{t})\,, \qquad m \geq 0, \tag{2.2}$$

where each S_i is either a concept or a role, $op_i \in \{\uplus, \cup\!\!\!\!-, \cap\!\!\!\!-\}$, p_i is a unary resp. binary predicate symbol, and $Q(\mathbf{t})$ is a dl-query. We call $p_1, \ldots, p_m$ its *input predicate symbols.* Intuitively, $op_i = \uplus$ (resp., $op_i = \cup\!\!\!\!-$) increases S_i (resp., $\neg S_i$) by the extension of p_i, while $op_i = \cap\!\!\!\!-$ constrains S_i to p_i.

A *dl-rule* r has the form

$$a \leftarrow b_1, \ldots, b_n, \mathrm{not}\ b_{n+1}, \ldots, \mathrm{not}\ b_m, \tag{2.3}$$

where any literal $b_1, \ldots, b_m \in B(r)$ may be a dl-atom. We define $H(r) = a$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_n\}$ and $B^-(r) = \{b_{n+1}, \ldots, b_m\}$. If $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then r is a *fact.* A *dl-program* $KB = (L, P)$ consists of a description logic knowledge base L and a finite set of dl-rules P.

The next example will illustrate our main ideas.

Example 2.7 ([Schindlauer, 2006])**.** An existing network must be extended by new nodes. The knowledge base L_N contains information about existing nodes and their interconnections as well as a definition of "overloaded" nodes (concept *HighTrafficNode*), which depends on the number of connections of the respective node (here, all nodes with more than three connections belong to *HighTrafficNode*):

[10]Additionally, in Chapter 3, we will extend these queries by the possibility to state conjunctive queries.

$\geq 1\ \mathit{wired} \sqsubseteq \mathit{Node};\ \ \top \sqsubseteq \forall \mathit{wired}.\mathit{Node};\ \ \mathit{wired} = \mathit{wired}^{-};$
$\geq 4\ \mathit{wired} \sqsubseteq \mathit{HighTrafficNode};$
$\mathit{Node}(n_1);\ \ \mathit{Node}(n_2);\ \ \mathit{Node}(n_3);\ \ \mathit{Node}(n_4);\ \ \mathit{Node}(n_5);$
$\mathit{wired}(n_1, n_2);\ \ \mathit{wired}(n_2, n_3);\ \ \mathit{wired}(n_2, n_4);$
$\mathit{wired}(n_2, n_5);\ \ \mathit{wired}(n_3, n_4);\ \ \mathit{wired}(n_3, n_5).$

To evaluate possible combinations of connecting the new nodes, the following program P_N is specified:

$$\mathit{newnode}(\mathit{add_1}). \tag{1}$$
$$\mathit{newnode}(\mathit{add_2}). \tag{2}$$
$$\mathit{overloaded}(X) \leftarrow \mathrm{DL}[\mathit{wired} \uplus \mathit{connect}; \mathit{HighTrafficNode}](X). \tag{3}$$
$$\mathit{connect}(X, Y) \leftarrow \mathit{newnode}(X), \mathrm{DL}[\mathit{Node}](X), \text{not } \mathit{overloaded}(Y), \text{not } \mathit{excl}(X, Y). \tag{4}$$
$$\mathit{excl}(X, Y) \leftarrow \mathit{connect}(X, Z), \mathrm{DL}[\mathit{Node}](Y), Y \neq Z. \tag{5}$$
$$\mathit{excl}(X, Y) \leftarrow \mathit{connect}(Z, Y), \mathit{newnode}(Z), \mathit{newnode}(X), Z \neq X. \tag{6}$$
$$\mathit{excl}(\mathit{add_1}, n_4). \tag{7}$$

Rules (1)–(2) define the new nodes to be added. Rule (3) imports knowledge about overloaded nodes in the existing network, taking new connections already into account. Rule (4) connects a new node to an existing one, provided the latter is not overloaded and the connection is not to be disallowed, which is specified by Rule (5) (there must not be more than one connection for each new node) and Rule (6) (two new nodes cannot be connected to the same existing one). Rule (7) states a specific condition: Node *add₁* must not be connected with n_4.

Two different semantics have been defined for dl-programs, the (strong) answer-set semantics [Eiter et al., 2004b] and the well-founded semantics [Eiter et al., 2004c]. The latter extends the well-founded semantics of [Van Gelder et al., 1991] to dl-programs. Well-founded semantics is based on the notion of greatest unfounded set and assigns a single three-valued model to every logic program. In addition, recent results define the well-founded semantics to a subclass of Hybrid MKNF KBs [Knorr et al., 2007]. In this work, we will extend the strong answer set semantics for dl-programs to our cq-programs in Section 3.

2.6 HEX-Programs

HEX-programs can be considered as successor to the dl-program formalism presented in Section 2.5. This new formalism generalizes dl-programs in two levels. First, the dl-atoms for querying external DL-KBs has been abstracted to accommodate a universal interface for arbitrary sources of external computation. This new transition point is formalised through the notion of external atoms and is comparable to various foreign function interfaces in Prolog interpreters like the SICStus Prolog C interface,[11] or the Foreign Language Interface of SWI-Prolog.[12] Contrary to the foreign function interfaces, the external atoms have been developed as a fully declarative framework with a model-theoretic semantics. Second, HEX-programs bring in support for higher-order reasoning, i.e., HEX-programs allow for specifying meta-reasoning tasks through higher order atoms. Both features have been

[11]http://www.sics.se/sicstus/
[12]http://www.swi-prolog.org/

developed to fit nicely into the answer-set programming paradigm, hence HEX-programs are a natural generalization of many proposed extensions in the ASP area.

2.6.1 Syntax of HEX-programs

Let $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ be mutually disjoint sets whose elements are called *constant names*, *variable names*, and *external predicate names*, respectively. Unless explicitly specified, elements from $\mathcal{X}$ (resp., $\mathcal{C}$) are denoted with first letter in upper case (resp., lower case), while elements from $\mathcal{G}$ are prefixed with "& ." We note that constant names serve both as individual and predicate names.

Elements from $\mathcal{C} \cup \mathcal{X}$ are called *terms.* A *higher-order atom* (or *atom*) is a tuple $(Y_0, Y_1, \ldots, Y_n)$, where $Y_0, \ldots, Y_n$ are terms; $n \geq 0$ is the *arity* of the atom. Intuitively, Y_0 is the predicate name, and we thus also use the more familiar notation $Y_0(Y_1, \ldots, Y_n)$. The atom is *ordinary*, if Y_0 is a constant.

For example, $(x, \mathit{rdf}{:}\mathit{type}, c)$, $\mathit{node}(X)$, and $D(a, b)$, are atoms; the first two are ordinary atoms.

An *external atom* is of the form

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m), \tag{2.4}$$

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two lists of terms (called *input* and *output* lists, respectively), and $\&g \in \mathcal{G}$ is an external predicate name. We assume that $\&g$ has fixed lengths $\mathit{in}(\&g) = n$ and $\mathit{out}(\&g) = m$ for input and output lists, respectively. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates.

Example 2.8 ([Eiter et al., 2005b]). The external atom $\&\mathit{reach}[\mathit{edge}, a](X)$ may be devised for computing the nodes which are reachable in the graph *edge* from the node a. Here, we have that $\mathit{in}(\&\mathit{reach}) = 2$ and $\mathit{out}(\&\mathit{reach}) = 1$.

A *rule* r is of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, \text{not } \beta_{n+1}, \ldots, \text{not } \beta_m, \tag{2.5}$$

where $m, k \geq 0$, $\alpha_1, \ldots, \alpha_k$ are atoms, and $\beta_1, \ldots, \beta_m$ are either atoms or external atoms. We define $H(r) = \{\alpha_1, \ldots, \alpha_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{\beta_1, \ldots, \beta_n\}$ and $B^-(r) = \{\beta_{n+1}, \ldots, \beta_m\}$. If $H(r) = \emptyset$ and $B(r) \neq \emptyset$, then r is a *constraint*, and if $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then r is a *fact*; r is *ordinary*, if it contains only ordinary atoms. Note that in contrast to dl-programs, HEX-programs allow for disjunctive heads and constraints.

A HEX-*program* is a finite set P of rules. It is *ordinary*, if all rules are ordinary.

2.6.2 Semantics of HEX-programs

We define the semantics of HEX-programs by generalizing the answer-set semantics by Gelfond and Lifschitz [1991]. To this end, we use the recent notion of a reduct as defined by Faber et al. [2004] (referred to as *FLP-reduct* henceforth) instead of to the traditional reduct by Gelfond and Lifschitz [1991]. The FLP-reduct admits an elegant and natural definition of answer sets for programs with aggregate atoms, since it ensures answer-set minimality, while the definition based on the traditional reduct lacks this important feature.

In the sequel, let P be a HEX-program. The *Herbrand base* of P, denoted HB_P, is the set of all possible ground versions of atoms and external atoms occurring in P obtained by replacing variables with constants from $\mathcal{C}$. The grounding of a rule r, $grnd(r)$, is defined accordingly, and the grounding of program P is given by $grnd(P) = \bigcup_{r \in P} grnd(r)$. Unless specified otherwise, $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ are implicitly given by P.

Example 2.9 ([Eiter et al., 2005b])**.** Given $\mathcal{C} = \{edge, arc, a, b\}$, ground instances of $E(X, b)$ are for instance $edge(a, b)$, $arc(a, b)$, $a(edge, b)$, and $arc(arc, b)$; ground instances of $\&reach[edge, N](X)$ are all possible combinations where N and $X \in \mathcal{C}$, for instance $\&reach[edge, edge](a)$, $\&reach[edge, arc](b)$, $\&reach[edge, edge](edge)$, etc.

An *interpretation relative to* P is any subset $I \subseteq HB_P$ containing only atoms. We say that I is a *model* of atom $a \in HB_P$, denoted $I \models a$, if $a \in I$.

With every external predicate name $\&g \in \mathcal{G}$, we associate an $(n+m+1)$-ary Boolean function $f_{\&g}$ assigning each tuple $(I, y_1 \ldots, y_n, x_1, \ldots, x_m)$ either 0 or 1, where $n = in(\&g)$, $m = out(\&g)$, $I \subseteq HB_P$, and $x_i, y_j \in \mathcal{C}$. We say that $I \subseteq HB_P$ is a *model* of a ground external atom $a = \&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$, denoted $I \models a$, if and only if $f_{\&g}(I, y_1 \ldots, y_n, x_1, \ldots, x_m) = 1$.

Note that in contrast to the semantics of higher-order atoms, which in essence reduces to first-order logic as customary (cf. [Ross, 1994]), the semantics of external atoms is in spirit of second order logic since it involves predicate extensions.

Example 2.10 ([Eiter et al., 2005b])**.** Let us associate with the external atom $\&reach$ a function $f_{\&reach}$ such that $f_{\&reach}(I, E, A, B) = 1$ iff B is reachable in the graph E from A. Let $I = \{e(b, c), e(c, d)\}$. Then, I is a model of $\&reach[e, b](d)$ since $f_{\&reach}(I, e, b, d) = 1$.

Let r be a ground rule. We define (i) $I \models H(r)$ iff there is some $a \in H(r)$ such that $I \models a$, (ii) $I \models B(r)$ iff $I \models a$ for all $a \in B^+(r)$ and $I \not\models a$ for all $a \in B^-(r)$, and (iii) $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$. We say that I is a *model* of a HEX-program P, denoted $I \models P$, iff $I \models r$ for all $r \in grnd(P)$. We call P *satisfiable*, if it has some model.

Given a HEX-program P, the *FLP-reduct* of P with respect to $I \subseteq HB_P$, denoted fP^I, is the set of all $r \in grnd(P)$ such that $I \models B(r)$. $I \subseteq HB_P$ is an *answer set of* P iff I is a minimal model of fP^I.

We next give an illustrative example.

Example 2.11 ([Eiter et al., 2005b])**.** Consider the following HEX-program P:

$$subRelation(brotherOf, relativeOf). \tag{1}$$
$$brotherOf(john, al). \tag{2}$$
$$relativeOf(john, joe). \tag{3}$$
$$brotherOf(al, mick). \tag{4}$$
$$invites(john, X) \vee skip(X) \leftarrow X \neq john, \&reach[relativeOf, john](X). \tag{5}$$
$$R(X, Y) \leftarrow subRelation(P, R), P(X, Y). \tag{6}$$
$$\leftarrow \&degs[invites](Min, Max), Min < 1. \tag{7}$$
$$\leftarrow \&degs[invites](Min, Max), Max > 2. \tag{8}$$

Informally, this program randomly selects a certain number of John's relatives for invitation. The first line states that *brotherOf* is a subrelation of *relativeOf*, and the next three lines give concrete facts. The disjunctive rule (5) chooses relatives, employing the external predicate $\&reach$ from Example 2.10. Rule (6) declares a generic subrelation inclusion exploiting higher-order atoms.

The constraints (7) and (8) ensure that the number of invitees is between 1 and 2, using (for illustration) an external predicate $\&degs$ from a graph library, where $f_{\&degs}(I, E, Min,$

Max) is 1 iff *Min* and *Max* is the minimum and maximum vertex degree of the graph induced by the edges E, respectively. As John's relatives are determined to be Al, Joe, and Mick, P has six answer sets, each of which contains one or two of the facts $\mathit{invites}(\mathit{john}, \mathit{al})$, $\mathit{invites}(\mathit{john}, \mathit{joe})$, and $\mathit{invites}(\mathit{john}, \mathit{mick})$.

In principle, the truth value of an external atom depends on its input and output lists and the entire model of the program. Practically however, we can identify certain types of input terms that allow to restrict the input interpretation to specific relations. The atom $\&\mathit{reach}[\mathit{edge}, a](X)$ for instance will only consider the extension of the predicate *edge* and the constant value a for computing its result and simply ignore the remaining interpretation. In Chapter we will formalize these two types of input terms and restrict the practical usage of external atoms to them, since such type information will support an efficient evaluation to a great extent.

We now state some basic properties of the semantics. The proofs for the next theorems can be found in [Schindlauer, 2006].

Theorem 2.12 ([Eiter et al., 2005b])**.** The answer-set semantics of HEX-programs extends the answer-set semantics of ordinary programs as defined by Gelfond and Lifschitz [1991], as well as the answer-set semantics of HiLog programs as defined by Ross [1994].

The next property, which is easily proved, expresses that answer sets adhere to the principle of minimality.

Theorem 2.13 ([Eiter et al., 2005b])**.** Every answer set of a HEX-program P is a minimal model of P.

A ground external atom a is called *monotonic relative to* P iff $I \subseteq I' \subseteq HB_P$ and $I \models a$ imply $I' \models a$. For instance, the ground versions of $\&\mathit{reach}[\mathit{edge}, a](X)$ are all monotonic.

Theorem 2.14 ([Eiter et al., 2005b])**.** Let P be a HEX-program without "not" and constraints. If all external atoms in $\mathit{grnd}(P)$ are monotonic relative to P, then P has some answer set. Moreover, if P is disjunction-free, it has a single answer set.

Notice that this property fails if external atoms can be non-monotonic. Indeed, we can easily model default negation $\mathrm{not}\, p(a)$ by an external atom $\&\mathrm{not}[p](a)$; the HEX-program $p(a) \leftarrow \&\mathrm{not}[p](a)$ amounts then to the ordinary program $p(a) \leftarrow \mathrm{not}\, p(a)$, which has no answer set.

2.7 Program Unfolding

This section introduces *Program Unfolding* as an optimization technique for logic programs. For this reason, we may apply the results of these procedure in the rewriting rules for optimization of cq-programs in Section 5.

We will begin with giving results and algorithms for term unification. This will then be used in partial deduction of disjunctive logic programs, which is the basis for our unfolding procedure of cq-programs in Algorithm 3.

2.7.1 Unification

In this section, we assume a first-order signature consisting of a set of *function symbols* $\mathcal{F}$ and a set of *variables* $\mathcal{V}$. *Terms* are build from $\mathcal{F}$ and $\mathcal{V}$ in the usual way and are denoted as $s, t, \ldots$, whereas variables are presented as X, Y, Z and so forth.

Definition 2.1 ([Baader and Snyder, 2001])**.** A *substitution*, denoted as σ, θ, η, and ρ, is a mapping from variables to terms which is almost everywhere equal to the identity. A substitution is represented as a function by a set of bindings of variables:

$$\{X_1/s_1, \ldots, X_n/s_n\}.$$

We say that a substitution is a *ground substitution*, if all s_i's are ground terms.

The application of a substitution σ to a term t, denoted $t\sigma$, is defined by induction on the structure of terms:

$$t\sigma := \begin{cases} X\sigma & \text{if } t = X \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

In case of $n = 0$, i.e., f is a constant symbol, $f\sigma = f$. Substitutions, too, may be applied to sets of terms, atoms, set of atoms, and rules in a logic program in the obvious fashion.

For a substitution σ, the *domain* is the set of variables

$$dom(\sigma) := \{X \mid X\sigma \neq X\}.$$

Composition of two substitutions is written $\sigma\theta$, and is defined by

$$t\sigma\theta = (t\sigma)\theta.$$

Example 2.15. Let $s = X$ and $t = c$ be terms, and let substitutions $\sigma = \{X/Y\}$ and $\theta = \{X/Z, Y/c\}$. Applying σ to s yields $s\sigma = Y$, whereas $t\sigma = c$. Moreover, $s\sigma\theta = t\sigma\theta = c$, but $s\theta \neq t\theta$.

Definition 2.2 ([Baader and Snyder, 2001])**.** Two substitutions are equal, denoted $\sigma = \theta$, if $X\sigma = X\theta$ for every variable $X \in \mathcal{V}$. We say that σ is *more general than* θ, if there exists an η such that $\theta = \sigma\eta$.

Example 2.16. Continuing our previous example, the substitutions $\sigma \neq \theta$ because $X\sigma \neq X\theta$. Moreover, let $p(X, X, Y)$ and $p(X', Y', Z)$ be atoms, and $\eta = \{X/Y', X'/Y', Y/Z\}$ and $\rho = \{X/W, X'/W, Y'/W, Y/Z\}$. Then

$$p(X, X, Y)\eta = p(X', Y', Z)\eta = p(Y', Y', Z)$$

and

$$p(X, X, Y)\rho = p(X', Y', Z)\rho = p(W, W, Z).$$

Additionally, η is more general than ρ, since $\rho = \eta\{Y'/W\}$.

Definition 2.3 ([Baader and Snyder, 2001])**.** A substitution σ is a *unifier* of two terms s and t if $s\sigma = t\sigma$; it is a *most general unifier* (or *mgu* for short), if for every unifier θ of s and t, σ is more general than θ.

Example 2.17. Let $s = g(X)$ and $t = g(g(Y))$ be first-order terms, then $\theta = \{X/g(Y)\}$ is an mgu for s and t.

In our answer-set programming setting, we do not have function symbols, hence there is no way to build arbitrarily nested terms. Algorithm 1 thus can be used as a procedure for computing the mgu for two terms s and t comprising only constant and variable symbols, which is a simpler version of the unification algorithm presented in [Baader and Snyder, 2001].

Definition 2.4. Let $p(\vec{X})$ and $p(\vec{Y})$ be two unifiable atoms with $\vec{X} = s_1, \ldots, s_n$ and $\vec{Y} = t_1, \ldots, t_n$. Then σ is an mgu of $p(\vec{X})$ and $p(\vec{Y})$, i.e., $p(\vec{X}\sigma) = p(\vec{Y}\sigma)$, where $\sigma = \sigma_{n+1}$ is the unifier obtained from $\sigma_1 = \emptyset$, and $\sigma_{i+1} = \mathit{unify}(s_i, t_i, \sigma_i)$, $1 \leq i \leq n$.

Example 2.18. The unifier η of Example 2.16 is an mgu for the atoms $p(X, X, Z)$ and $p(X', Y', Z)$.

Algorithm 1: *unify*(s, t, σ): Unification without function symbols

```
Input: Function-free terms $s$ and $t$, unifier $\sigma$
Result: mgu of $s$ and $t$ or failure
if $s$ is a variable then $s \leftarrow s\sigma$
if $t$ is a variable then $t \leftarrow t\sigma$
if $s$ is a variable and $s = t$ then return $\sigma$
else if $s = c_1$ and $t = c_2$ then
    if $c_1 = c_2$ then return $\sigma$
    else Exit with failure
else if $s$ is not a variable then return $\sigma\{t/s\}$
return $\sigma\{s/t\}$
```

2.7.2 Partial deduction of disjunctive logic programs

Partial deduction of disjunctive logic programs (or unfolding of disjunctive rules), as shown in [Sakama and Seki, 1994, 1997], is known as a optimization technique for logic programs; in fact, it is a useful procedure for optimizing abductive logic programs and compiling propositional disjunctive programs. Moreover, in [Eiter et al., 2004a] it is shown that *partial evaluation*, which is another name for partial deduction, is useful for simplifying logic programs.

By performing partial deduction on a part of a given logic program, we might save some fixpoint iterations while evaluating a program. As will be shown in Section 5, we are particularly interested in the resolvent r' of $\pi_{r;a(\vec{Y})}(P)$, which ultimately brings (by repetitive application of the partial deduction technique) special types of atoms into the body of a single rule. This will become very useful for dl-atoms, which were defined previously and will be extended later on. When such dl-atoms occur in a body, we apply further optimization methods to this newly created rule, which were impossible in the original program without partial deduction applied.

Definition 2.5 ([Sakama and Seki, 1997])**.** Let P be a disjunctive logic program and let r be a rule in P of the form

$$r : H \leftarrow a(\vec{Y}), B.$$

Suppose that $r_1, \ldots, r_l$ are all of the rules in P such that

$$r_i : a(\vec{Y_i}) \vee H_i \leftarrow B_i \qquad (1 \leq i \leq l),$$

where $a(\vec{Y_i}\theta_i) = a(\vec{Y}\theta_i)$ holds with an mgu θ_i for each i.

Then a *disjunctive partial deduction* of P (with respect to r on a') is defined as a *residual program* $\pi_{r;a(\vec{Y})}(P)$ such that

$$\pi_{r;a(\vec{Y})}(P) = \begin{cases} P \cup \{r'_1, \ldots, r'_l\} & \text{if there is a rule } r_i \in P \text{ such that } H_i \\ & \text{contains an atom unifiable with } a(\vec{Y}) \\ (P \setminus \{r\}) \cup \{r'_1, \ldots, r'_l\} & \text{otherwise,} \end{cases}$$

where

$$r'_i : (H \vee H_i \leftarrow B, B_i)\,\theta_i.$$

Example 2.19 ([Sakama and Seki, 1997])**.** Let P be the program

$$r_1 : p(a) \vee p(b) \vee q(c).$$
$$r_2 : p(X) \leftarrow q(X).$$
$$r_3 : r(Y) \leftarrow p(Y).$$

The disjunctive partial deduction $\pi_{r_3;p(Y)}(P)$ is the program $P \cup \{r'_1, r'_2\}$, where

$$\begin{aligned} &r'_1 : r(a) \vee p(b) \vee q(c). && \text{(by } r_3 \text{ and } r_1 \text{ with the mgu } \{Y/a\}) \\ &r'_2 : r(X) \leftarrow q(X). && \text{(by } r_3 \text{ and } r_2 \text{ with the mgu } \{Y/X\}) \end{aligned}$$

On the other hand, $\pi_{r_2;q(X)}(P) = \{r_1, r_3, r''_1\}$, where

$$r''_1 : p(a) \vee p(b) \vee p(c). \qquad \text{(by } r_2 \text{ and } r_1 \text{ with the mgu } \{X/c\})$$

The next results justify that partial deduction preserves the semantics of a given disjunctive program P. First, we setup a Lemma which shows that answer sets are "minimal" with respect to disjunctive heads.

Lemma 2.20 ([Sakama and Seki, 1997])**.** Let P be a positive disjunctive program and I its minimal model. Then a ground atom a is in I iff there is a ground rule $a \vee H \leftarrow B$ from P such that $I \setminus \{a\} \models B$ and $I \setminus \{a\} \not\models H$.

Next, we give the main result of partial deduction. In the following, let $MM(P)$ (resp. $SM(P)$) denote the minimal (resp. stable) models of P, and $\pi(P)$ be any residual program of P.

Theorem 2.21 ([Sakama and Seki, 1997])**.** Let P be a positive (resp. normal) disjunctive program. Then $MM(P) = MM(\pi(P))$ (resp. $SM(P) = SM(\pi(P))$).

Partial deduction of disjunctive programs and its results will be generalized to dl- and cq-programs in Chapter 5. By the results of Chapter 4, we apply our unfolding mechanism to HEX-programs with so called DL external atoms.

2.8 Conjunctive Queries

Conjunctive queries (CQs) and union of conjunctive queries (UCQs) over relational databases were introduced in [Chandra and Merlin, 1977]. They are equivalent to select-project-join database queries and are the formal basis of SQL. For a thorough introduction on query languages cf. [Abiteboul et al., 1995]. In the following, we assume some understanding of CQs over relational databases.

Informally, such queries allow for expressing a conjunction of atoms over the database, such that each tuple in the set of answers of the CQ is a satisfying assignment for the conjunction. Take, for instance, the following CQ[13] q

$$q(X, Y) \leftarrow P(X), R(Y, Z), S(X, Y, Z),$$

where the relations in the body of q are defined over a particular database schema and the predicate name q does not appear as relation in this schema. Given a database instance over that schema, we are in a position to answer q and get all pairs $\langle X, Y\rangle$ satisfying the body atoms, while the outcome of Z is not exported. We call X and Y the distinguished variables of q, i.e., they will "carry" the answers to q. Z is called a non-distinguished variable with the intuitive meaning that Z is existentially quantified in q, i.e., in a particular answer tuple to q, Z must contain the same constant in the atoms $R(Y, Z)$ and $S(X, Y, Z)$, but we do not care which value this might be. This sort of problem in CQs is called query answering, the problem to compute all answers to a given CQ and a given database.

[13]This query is in datalog syntax. Note that one can define the syntax of CQs in many equivalent ways.

Chandra and Merlin, too, studied the query containment problem of CQs. Query containment of two CQs q_1 and q_2, formally $q_1 \subseteq q_2$, decides whether the answers to q_1 are a subset of the answers to q_2 over a database instance. If $q_1 \subseteq q_2$ and $q_2 \subseteq q_1$, then q_1 is equivalent to q_2 (denoted $q_1 \equiv q_2$). Equivalence of CQs is crucial for query optimization in the database area. In addition, query containment is tightly coupled to query answering problem.

Note that CQs in the database field are evaluated in an active domain, i.e., under closed world assumption. The database is considered complete (see also Section 1.3). Take, for instance, the following first-order query

$$q = \{X \mid \forall Y R(X,Y)\}$$

over a database instance $R = \{\langle 1,1\rangle\}$ and the domain $\mathbf{D} = \{1,2\}$. Using this domain, $q = \emptyset$, since $R(1,2)$ does not hold. We call such queries unsafe, since the evaluation depends on the supplied domain. However, if we restrict the domain to the individuals from the database instance, query answering does not depend on the domain anymore, thus quantified variables range over named individuals from the domain. In this case, $\mathbf{D} = \{1\}$ and $q = \{1\}$. In this setting, queries are considered safe. Nevertheless, in description logics, this assumption is not present, in fact, CQ answering is tightly linked to CQ answering over incomplete databases, cf. the excellent survey [van der Meyden, 1998].

2.8.1 Conjunctive queries over description logics

In our setting, we use CQs and UCQs over description logics knowledge bases. The reasoning problems that arise in (U)CQs are a natural continuation of the traditional reasoning tasks presented in Section 2.3.

The next definition gives the syntax of (U)CQs over DL-KBs. Note that we use another syntax for (U)CQs in our cq-programs in Section 3 to accommodate the syntax of queries in dl-atoms.

Definition 2.6. A *conjunctive query* $q(\vec{X})$ is an expression of form

$$\exists \vec{Y} Q_1(\vec{X}_1) \wedge \cdots \wedge Q_n(\vec{X}_n),$$

where for $1 \leq i \leq n$ each Q_i is a concept or role expression over a DL-KB, $\vec{X}_i$ is a singleton or pair of variables or individuals if Q_i is a concept or role expression, resp., $\vec{X} \subseteq \bigcup_{i=1}^{n} \mathrm{vars}(X_i)$ are called *distinguished* (or *output*) *variables*, $\vec{Y} \subseteq \bigcup_{i=1}^{n} \mathrm{vars}(X_i)$ are called *non-distinguished* (or *existential*) *variables*, and $\vec{X}$ and $\vec{Y}$ do not share variables.

A *union of conjunctive queries* $q(\vec{X})$ is an expression of form

$$q_1(\vec{X}) \vee \cdots \vee q_m(\vec{X}),$$

where for $1 \leq i \leq m$ each $q_i(\vec{X})$ is a conjunctive query, and $\vec{X}$ are the *distinguished variables* of $q(\vec{X})$.

A (union of) conjunctive queries $q(\vec{X})$ is a *Boolean (Union of) Conjunctive Queries*, if there are no distinguished variables. We denote boolean (U)CQs by q.

In the following, we define central reasoning problems for (U)CQs over description logics.

Definition 2.7. Given a (U)CQ q without distinguished variables (or *boolean* (U)CQ) over L , we say that L *entails* q, denoted $L \models q$, iff $\mathcal{I} \models L$ implies $\mathcal{I} \models q$ for every interpretation $\mathcal{I}$. Deciding $L \models q$ is the problem called *query entailment.*

Given a (U)CQ $q(\vec{X})$ over L with n distinguished variables $\vec{X}$ and let $\vec{c} = \langle c_1, \ldots, c_n \rangle$ be an n-tuple of individuals. The boolean (U)CQ $q(\vec{c})$ is obtained from $q(\vec{X})$ by replacing each occurrence of a distinguished variable X_i by c_i $(1 \leq i \leq n)$. We say that $\vec{c}$ is an *answer for* $q(\vec{X})$ if $L \models q(\vec{c})$. The *answers of* $q(\vec{X})$ *over* L are all such possible n-tuples $\vec{c}$ which satisfy $L \models q(\vec{c})$.

Example 2.22. Consider the DL knowledge base L in Example 2.5 by an extended version L', which consists of following additional axioms:

$$\begin{aligned} \mathit{fatherOf} &\sqsubseteq \mathit{parentOf} && (9)\\ \mathit{motherOf} &\sqsubseteq \mathit{parentOf} && (10)\\ \mathit{Female} \sqcup \mathit{Male} &\sqsubseteq \exists \mathit{parentOf}^{-}.(\mathit{Female} \sqcup \mathit{Male}) && (11)\\ \forall \mathit{parentOf}.\mathit{SpaceTourist} &\sqsubseteq \mathit{SteadyNerves} && (12) \end{aligned}$$

Axiom (9) and (10) states that each father and each mother is a parent, resp. In axiom (11) we declare that all men and women have a parent which is a man or a woman. Additionally, (12) states that all parents of spacetourists have steady nerves.

Now consider the conjunctive query

$$q_1(X) = \exists Y, Z\, \mathit{parentOf}(X, Y) \land \mathit{SteadyNerves}(X) \land \mathit{parentOf}(Z, X)$$

over L'. The answers to this query are *Fakhri* and *Rick*, since the two of them are members of *SteadyNerves* as they are parents of the spacetourists *Anousheh* and *Mark*, resp., and by (11), they have some unnamed parents.

The conjunctive query

$$q_2(X) = \exists Y\, \mathit{motherOf}(Y, X) \land \mathit{SpaceTourist}(X)$$

asks for all spacetourists with a mother. $L' \models q_2(\mathit{Anousheh})$ holds, but $L' \not\models q_2(\mathit{Mark})$, as L' only states that every person has a parent (which could be a mother or a father).

Definition 2.8. Given (U)CQs q_1 and q_2, the *query containment* problem is to decide whether $L \models q_2$ if $L \models q_1$ for every DL-KB L.

As shown in [Abiteboul and Duschka, 1998, Calvanese et al., 2007b], query containment can be used to answer queries and vice versa.

A first study on these problems has been introduced in [Levy and Rousset, 1998] and [Calvanese et al., 1998], which shows that query containment (and hence CQ answering) is decidable in one of the most expressive DLs called $\mathcal{DLR}_{reg}$. This results are very important for this work, since they show that (i) answering (U)CQs over very expressive DLs is in fact decidable, thus rendering our cq-programs decidable, and (ii) they give hints on the worst case complexity of answering (U)CQs, which can serve as a basis for studying complexity issues in our new formalism too.

Recent results show that CQ containment and answering is decidable for very expressive DLs like $\mathcal{SHIQ}$ [Ortiz de la Fuente et al., 2006a, Glimm et al., 2007a] and $\mathcal{SHOQ}$ [Glimm et al., 2007b]. These results are fundamental, since they provide decidability results for the closely related DLs $\mathcal{SHIF}$ and parts of $\mathcal{SHOIN}$, hence rendering CQ answering in OWL Lite and parts of OWL DL (without concrete domain $\mathbf{D}$ and inverse roles) decidable. Recent efforts in the DL community include finding algorithms for CQ answering in $\mathcal{SHOIQ}$—at the time of writing, no results have been published. This would close the case for CQ answering in $\mathcal{SHOIN}$, since $\mathcal{SHOIN}$ is a restricted form of $\mathcal{SHOIQ}$. Moreover, the even more expressive two-way regular path queries, a generalization of UCQs, have been shown decidable for the DL $\mathcal{ALCQIb}_{reg}$ in [Calvanese et al., 2007a]. On the other end of the scale, simple DLs like the DL-Lite [Calvanese et al., 2005] or the $\mathcal{EL}$ [Rosati, 2007a, Krötzsch and Rudolph, 2007, Krisnadhi and Lutz, 2007] family of DLs have been shown tractable (i.e., L-hard for DL-Lite and P-complete for $\mathcal{EL}$) with respect to data complexity. An important result for the upcoming OWL 1.1 standard is in [Krötzsch et al., 2007], which shows that conjunctive query answering in unrestricted $\mathcal{EL}^{++}$—a tractable fragment of the DL $\mathcal{SROIQ}$—is undecidable, whereas under certain restrictions, CQ answering in $\mathcal{EL}^{++}$ is in fact decidable. The concrete complexity results for deciding the CQ answering problem for Horn-$\mathcal{SHIQ}$ [Hustadt et al., 2005], which is a tractable fragment of $\mathcal{SHIQ}$ and thus $\mathcal{SROIQ}$, are still open. See also [Grau et al., 2006] for a complete list of tractable fragments of $\mathcal{SROIQ}$.

In [Rosati, 2007b] some fundamental limits of CQ answering have been presented. While plain CQs and UCQs without (in)equalities and negation are in fact decidable for a broad range of DLs, extending CQs and UCQs by allowing inequality atoms will in fact render CQ answering undecidable for $\mathcal{AL}$, $\mathcal{ALC}$, and $\mathcal{ALCHIQ}$, whereas UCQ answering with inequality is undecidable starting from DL-Lite$_R$ on. Similarly, conjunctive query answering becomes undecidable for certain more expressive DLs when we allow negation in the query atoms. This valuable results are of greatest importance for our rewriting rules in Section 5, since we cannot apply all of them in every description logic without losing decidability of (U)CQ answering.

For this is the game of coupling.

—Sally in Coupling, *Bed Time*

3 cq-Programs: Extending dl-Atoms by Conjunctive Queries

This chapter deals with the first of our main contributions, the extension of dl-programs by allowing (union of) conjunctive queries in dl-atoms and disjunctive rules in logic programs (cf. Section 2.5). We define formal semantics for cq-programs by providing suitable extensions to the strong answer-set semantics for dl-programs. Moreover, we comment on the richer expressiveness of cq-programs compared to other formalisms used for combining logic programming and DLs, and assert that our extension retains decidability, whenever (U)CQ answering of the underlying DL is decidable.

3.1 Introduction

Rule formalisms that combine logic programming with other sources of knowledge, especially terminological knowledge expressed in Description Logics (DLs), have gained increasing interest in the past years. This process was mainly fostered by current efforts in the Semantic Web development of designing a suitable rules layer on top of the existing ontology layer. Such couplings between DLs (in the form of ontologies) and logic programming appear in different flavors, which roughly can be categorized in (i) systems with strict semantic integration and (ii) systems with strict semantic separation, which amounts to coupling heterogeneous systems [Rosati, 2006b, Eiter et al., 2006a, Antoniou et al., 2005, Pan et al., 2004].

In this work, we will concentrate on the latter, considering ontologies as an external source of information with semantics treated independently from the logic program. One representative of this category was presented in [Eiter et al., 2004b, 2006a], extending the answer set semantics towards so-called *dl-programs*. They consist of a DL part L and a rule part P, and where queries from P to L are allowed. These queries are facilitated by a special type of atoms which also permit to hypothetically enlarge the assertional part of L with facts imported from the logic program P, thus allowing for a bidirectional flow of information.

The types of queries expressible by dl-atoms in [Eiter et al., 2004b, 2006a] are concept and role membership queries, as well as subsumption queries. Since the semantics of logic programs is usually defined over a domain of explicit individuals, this approach may fail to derive certain consequences, which are implicitly contained in the DL-KB L.

Example 3.1. Consider this simplified version of an example from [Motik et al., 2005]:

$$L = \left\{ \begin{array}{c} \mathit{father} \sqsubseteq \mathit{parent}, \exists \mathit{father}.\exists \mathit{father}^{-}.\{\mathit{Remus}\}(\mathit{Romulus}), \\ \mathit{hates}(\mathit{Cain}, \mathit{Abel}), \mathit{hates}(\mathit{Romulus}, \mathit{Remus}), \\ \mathit{father}(\mathit{Cain}, \mathit{Adam}), \mathit{father}(\mathit{Abel}, \mathit{Adam}) \end{array} \right\}$$

$$P = \{\mathit{BadChild}(X) \leftarrow \mathrm{DL}[\mathit{parent}](X, Z), \mathrm{DL}[\mathit{parent}](Y, Z), \mathrm{DL}[\mathit{hates}](X, Y).\}$$

Apart from the explicit facts, L states that each *father* is also a *parent* and that Romulus and Remus have a common father. The single rule in P specifies that an individual hating a sibling is a *BadChild*. From this dl-program, $\mathit{BadChild}(\mathit{Cain})$ can be concluded, but not $\mathit{BadChild}(\mathit{Romulus})$, though it is implicitly stated that he and *Remus* have the same father.

The reason is that, in a dl-program, variables must be instantiated over its Herbrand base (containing the individuals in L and P), and thus unnamed individuals like the father of Romulus and Remus, are not considered. In essence, this means that dl-atoms only allow for building conjunctive queries that are *DL-safe* in the spirit of [Motik et al., 2005], which ensures that all variables in the query can be instantiated to named individuals. While this was mainly motivated for retaining decidability of the formalisms, unsafe conjunctive queries are admissible under specific conditions [Rosati, 2006b]. In this vein, we extend dl-programs by permitting conjunctive queries or unions thereof to L as first-class citizens in the language.

Example 3.2. In our Example 3.1, we may use

$$P' = \{\mathit{BadChild}(X) \leftarrow \mathrm{DL}[\mathit{parent}(X, Z), \mathit{parent}(Y, Z), \mathit{hates}(X, Y)](X, Y).\}$$

where the body of the rule is a CQ $\{\mathit{parent}(X, Z),\ \mathit{parent}(Y, Z),\ \mathit{hates}(X, Y)\}$ over L with distinguished variables X and Y. Then we shall obtain the desired result, that $\mathit{BadChild}(\mathit{Romulus})$ is concluded.

The extension of dl-programs to cq-programs, introduced in [Eiter et al., 2007a], has some attractive features.

• First and foremost, the expressiveness of the formalism is increased significantly, since existentially quantified and therefore unnamed individuals can be respected in query answering through the devices of cq-atom and ucq-atom.

• In addition, cq-programs have the nice feature that the integration of rules and the ontology is decidable whenever answering CQs resp. UCQs over the ontology (possibly extended with assertions) is decidable. In particular, recent results on the decidability of answering CQs and UCQs for expressive DLs can be exploited in this direction [Ortiz de la Fuente et al., 2006b,a, Glimm et al., 2007a]. Furthermore, it allows to express, via conjunction of cq-atoms and negated cq-atoms in rule bodies, certain decidable conjunctive queries which negations; note that negation leads quickly to undecidability [Rosati, 2007b].

• The availability of conjunctive queries opens the possibility to express joins in different, equivalent ways.

Example 3.3. Both

$$r : \mathit{BadParent}(Y) \leftarrow \mathrm{DL}[\mathit{parent}](X, Y), \mathrm{DL}[\mathit{hates}](Y, X)$$

and

$$r' : \mathit{BadParent}(Y) \leftarrow \mathrm{DL}[\mathit{parent}(X, Y), \mathit{hates}(Y, X)](X, Y)$$

equivalently single out (not necessarely all) bad parents. Here, in r the join between *parent* and *hates* is performed in the logic program, while in r' it is performed on the DL-side. Note that the queries in r and r' are DL-safe and involve only distinguished variables in the CQ, resp., hence no unnamed individuals are taken into account during query evaluation.

Since DL-reasoners including RacerPro, KAON2, and Pellet increasingly support answering CQs, this can be exploited to push joins from the rule part to the DL-reasoner, but also vice versa. Since calls to the DL-reasoner are an inherent bottleneck in evaluating cq-programs, in this way the performance can be significantly improved.

The last characteristic allows for optimizing cq-programs by means of the rewriting rules presented in Chapter 5. The experimental prototype for cq-programs is ready for use and explained in more detail in the Implementation chapter. To our knowledge, it is currently the most expressive implementation of integrating nonmonotonic rules and ontologies.

3.2 Syntax of cq-Programs

We assume familiarity with description logics (DLs) (cf. [Baader et al., 2003]), in particular $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$.[1] A DL-KB L is a finite set of axioms in the respective DL. We denote logical consequence of an axiom α from L by $L \models \alpha$.

As in [Eiter et al., 2004b, 2006a], we assume a function-free first-order vocabulary Φ of nonempty finite sets $\mathcal{C}$ and $\mathcal{P}$ of constant resp. predicate symbols, and a set $\mathcal{X}$ of variables. As usual, a *classical literal* (or *literal*), l, is an atom a or a negated atom $\neg a$.

Informally, a cq-program consists of a DL-KB L and a generalized disjunctive program P, which may involve queries to L. Roughly, such a query may ask whether a specific description logic axiom, a conjunction or a union of conjunctions of DL axioms is entailed by L or not.

We first define dl-queries, which is one form to express queries to the description logic knowledge base L. A *dl-query* $Q(\vec{t})$ is either

– a concept inclusion axiom F or its negation $\neg F$, or

– of the forms $C(t)$ or $\neg C(t)$, where C is a concept and t is a term, or

– of the forms $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, where R is a role and t_1, t_2 are terms.

A *conjunctive query* (CQ) $q(\vec{X})$ is an expression

$$\{\vec{X} \mid Q_1(\vec{X_1}), Q_2(\vec{X_2}), \ldots, Q_n(\vec{X_n})\}, \tag{3.1}$$

where $n \geq 0$, each Q_i is a concept or role expression and each $\vec{X_i}$ is a singleton or pair of variables and individuals matching the arity of Q_i, and where $\vec{X} \subseteq \bigcup_{i=1}^{n} \text{vars}(\vec{X_i})$ are its *distinguished* (or *output*) variables.

A *union of conjunctive queries* (UCQ) $q(\vec{X})$ is an expression of form

$$\{\vec{X} \mid q_1(\vec{X}) \vee \cdots \vee q_m(\vec{X})\} \tag{3.2}$$

of CQs $q_i(\vec{X})$ for $m \geq 0$.

Intuitively, a CQ $q(\vec{X})$ is a conjunction $Q_1(\vec{X_1}) \wedge \cdots \wedge Q_n(\vec{X_n})$ of concept and role expressions with possibly existential quantified variables, which is true if all conjuncts are satisfied. A UCQ $q(\vec{X})$ is satisfied, whenever some $q_i(\vec{X})$ is satisfied. We will omit the output variables $\vec{X}$ from CQs and UCQs if it is clear from the context, especially when (U)CQs are used in dl-atoms. Here, the output of the dl-atom and of the (U)CQ are equal.

[1] We focus on these DLs because they underlie OWL-Lite and OWL-DL. Conceptually, cq-programs can be defined for other DLs as well.

Example 3.4. In our Example 3.2, $cq_1(X,Y) = \{X,Y \mid \mathit{parent}(X,Z),\ \mathit{parent}(Y,Z),\ \mathit{hates}(X,Y)\}$ and $cq_2(X,Y) = \{X,Y \mid \mathit{father}(X,Y),\ \mathit{father}(Y,Z)\}$ are CQs with output X,Y, and $\mathit{ucq}(X,Y) = cq_1(X,Y) \vee cq_2(X,Y)$ is a UCQ.

A dl-atom is of form

$$\mathrm{DL}[\lambda; q](\vec{X}), \tag{3.3}$$

where $\lambda = S_1\ op_1\ p_1, \ldots, S_m\ op_m\ p_m$ $(m \geq 0)$ is a list of expressions $S_i\ op_i\ p_i$ called *input list*, each S_i is either a concept or a role, $op_i \in \{\uplus, \cup\!\!\!-, \cap\!\!\!-\}$, p_i is a predicate symbol matching the arity of S_i, and q is a (U)CQ with output variables $\vec{X}$ (in this case, (3.3) is called a *(u)cq-atom*), or $q(\vec{X})$ is a dl-query. Each p_i is an *input predicate symbol*; intuitively, $op_i = \uplus$ increases S_i by the extension of p_i, while $op_i = \cup\!\!\!-$ increases $\neg S_i$; $op_i = \cap\!\!\!-$ constrains S_i to p_i.

Example 3.5. The cq-atom $\mathrm{DL}[\mathit{parent} \uplus p; \mathit{parent}(X,Y), \mathit{parent}(Y,Z)](X,Z)$ with output X,Z extends L by adding the extension of p to *parent*, and then joins *parent* with itself.

A *cq-rule* r is of the form

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \mathrm{not}\, b_{m+1}, \ldots, \mathrm{not}\, b_n, \tag{3.4}$$

where every a_i is a literal and every b_j is either a literal or a dl-atom. We define $H(r) = \{a_1, \ldots, a_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$. If $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then r is a *fact*. If $H(r) = \emptyset$ and $B(r) \neq \emptyset$, then r is a *constraint*. We denote by $B^+_{dl}(r)$ (resp., $B^-_{dl}(r)$) the set of all dl-atoms occurring in $B^+(r)$ (resp., $B^-(r)$). A *cq-program* $KB = (L,P)$ consists of a DL-KB L and a finite set of cq-rules P.

Example 3.6. In the introduction, the pairs (L,P) and (L,P') are different versions of a cq-program for determining bad children over a knowledge base.

The following program is more involved, and uses nonmonotonic negation.

Example 3.7. Let $KB = (L,P)$, where L is the well-known wine ontology[2] and P is as follows:

$$\begin{aligned}
\mathit{visit}(L) \vee \neg\mathit{visit}(L) \leftarrow{}& \mathrm{DL}[\mathit{WhiteWine}](W), \mathrm{DL}[\mathit{RedWine}](R), && (1)\\
& \mathrm{DL}[\mathit{locatedIn}](W,L), \mathrm{DL}[\mathit{locatedIn}](R,L), &&\\
& \mathrm{not}\,\mathrm{DL}\big[\mathit{locatedIn}(L,L')\big](L). &&\\
\leftarrow{}& \mathit{visit}(X), \mathit{visit}(Y), X \neq Y. && (2)\\
\mathit{some\_visit} \leftarrow{}& \mathit{visit}(X). && (3)\\
\leftarrow{}& \mathrm{not}\ \mathit{some\_visit}. && (4)\\
\mathit{delicate\_region}(W) \leftarrow{}& \mathit{visit}(L), \mathit{delicate}(W), \mathrm{DL}[\mathit{locatedIn}](W,L). && (5)\\
\mathit{delicate}(W) \leftarrow{}& \mathrm{DL}[\mathit{hasFlavor}](W, \mathit{wine{:}Delicate}). && (6)
\end{aligned}$$

Informally, rule (1) selects a maximal region in which both red and white wine grow, and the next three rules (2)–(4) make sure that exactly one such region is picked, by enforcing that no more than two regions are chosen (rule (2)) and that at least one is chosen (rule (3) and (4)). The last two rules (5) and (6) single out all the sub-regions of the selected region producing some delicate wine, i.e., if a wine has a delicate flavor which is specified by individual *wine:Delicate*. Figure 3.1 displays the relevant concepts and roles in an out-take of the wine ontology class hierarchy.

[2] `http://www.w3.org/TR/owl-guide/wine.rdf`

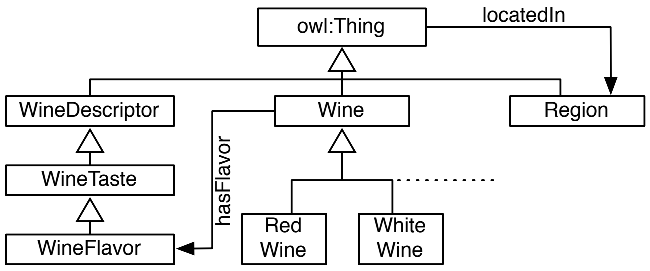

Figure 3.1: Wine ontology hierarchy

Note that P uses only—with one exception in (1)—instance retrieval queries. The weakly negated dl-atom in rule (1) is a conjunctive query with only one query atom, since we have to remove the non-distinguished variable L' from the output to keep the rule safe. The program will be used throughout the paper for demonstrating our rewriting methods.

3.3 Semantics of cq-Programs

We first recall the semantics of CQs and UCQs on a DL knowledge base L.

For any conjunctive query $q(\vec{X}) = \{\vec{X} \mid Q_1(\vec{X}_1), Q_2(\vec{X}_2), \ldots, Q_n(\vec{X}_n)\}$, let

$$\phi_q(\vec{X}) = \exists \vec{Y} \bigwedge_{i=1}^{n} Q_i(\vec{X}_i),$$

where $\vec{Y}$ are the variables not in $\vec{X}$, and for any union of conjunctive queries $q(\vec{X}) = \{\vec{X} \mid q_1(\vec{X}) \vee \cdots \vee q_m(\vec{X})\ \}$, let

$$\phi_q(\vec{X}) = \bigvee_{i=1}^{m} \phi_{q_i}(\vec{X}).$$

Then, for any (U)CQ $q(\vec{X})$, the set of *answers of* $q(\vec{X})$ *on* L is the set of tuples

$$ans(q(\vec{X}), L) = \{\vec{c} \in \mathcal{C}^{|\vec{X}|} \mid L \models \phi_q(\vec{c})\}.$$

Example 3.8. The CQ $cq_1(X, Y)$ from Example 3.4 has on L from Example 3.1 the set of answers $ans(cq_1(X, Y), L) = \{\langle Cain, Abel\rangle\}$.

Let $KB = (L, P)$ be a cq-program. The *Herbrand base* of P, denoted HB_P, is the set of all ground literals with a standard predicate symbol that occurs in P and constant symbols in $\mathcal{C}$. An *interpretation* I relative to P is a consistent subset of HB_P. We say I is a *model* of $l \in HB_P$ under L, or I *satisfies* l under L, denoted $I \models_L l$, iff $l \in I$.

A ground dl-atom $a = \mathrm{DL}[\lambda; Q](\vec{c})$ is *satisfied w.r.t.* I, denoted $I \models_L a$, if $L \cup \lambda(I) \models Q(\vec{c})$, where $\lambda(I) = \bigcup_{i=1}^{m} A_i(I)$ and

- $A_i(I) = \{S_i(\vec{e}) \mid p_i(\vec{e}) \in I\}$, for $op_i = \uplus$;
- $A_i(I) = \{\neg S_i(\vec{e}) \mid p_i(\vec{e}) \in I\}$, for $op_i = \cup\!\!\!-$;
- $A_i(I) = \{\neg S_i(\vec{e}) \mid p_i(\vec{e}) \in I$ does not hold$\}$, for $op_i = \cap\!\!\!-$.

Now, given a ground instance $a(\vec{c})$ of a (u)cq-atom $a(\vec{X}) = \mathrm{DL}[\lambda; q](\vec{X})$, i.e., all output variables in $q(\vec{X})$ are replaced by constants, we say that I *satisfies* $a(\vec{c})$, denoted $I \models_L a(\vec{c})$, if $\vec{c} \in ans(q(\vec{X}), L \cup \lambda(I))$.

Let r be a ground cq-rule. We define (i) $I \models_L H(r)$ iff there is some $a \in H(r)$ such that $I \models_L a$, (ii) $I \models_L B(r)$ iff $I \models_L a$ for all $a \in B^+(r)$ and $I \not\models_L a$ for all $a \in B^-(r)$, and (iii) $I \models_L r$ iff $I \models_L H(r)$ whenever $I \models_L B(r)$. We say that I is a *model* of a cq-program $KB = (L, P)$, or I *satisfies* KB, denoted $I \models KB$, iff $I \models_L r$ for all $r \in \mathit{ground}(P)$. We say KB is *satisfiable* (resp., *unsatisfiable*) iff it has some (resp., no) model. The (strong) answer sets of KB, which amount to particular models of KB, will be addressed below. The strong answer-set semantics for cq-programs is a quite straightforward extension to the strong answer-set semantics for dl-programs. This is done by moving from the least model semantics of positive dl-programs [Eiter et al., 2004b] to the minimal model semantics of positive cq-programs. Disjunctive rules were already considered in [Eiter et al., 2006a], but only for ordinary monotonic dl-atoms, i.e., dl-atoms with $\uplus$ and $\cup\!\!\!\!-$ operators; we provide semantics for cq- and ucq-atoms with the full range of operators. Moreover, the minimal models of the extended version of the strong dl-reduct will then be identified as the strong answer sets of KB. Extending weak answer-set semantics [Eiter et al., 2004b] for dl-programs to cq-programs or well-founded semantics [Eiter et al., 2004c] can be imagined, but this is not considered here.

3.3.1 Minimal-model semantics for positive cq-programs

We first consider positive cq-programs, i.e., cq-programs (L, P) without $\cap\!\!\!\!-$ in the input lists of every dl-atom occurring in P and with $B^-(r) = \emptyset$ for all $r \in P$. Like for ordinary positive programs, every nondisjunctive positive cq-program which is satisfiable has a single minimal model, which naturally characterizes its semantics.

Lemma 3.9. Let $KB = (L, P)$ be a normal positive cq-program. If the interpretations $I_1, I_2 \subseteq HB_P$ are models of KB, then $I_1 \cap I_2$ is also a model of KB.

Proof. The proof follows the line of the proof for Lemma 4.2 in [Eiter et al., 2007b]. □

As an immediate corollary of this result, every satisfiable positive normal cq-program KB has a unique minimal model, denoted M_{KB}, which is contained in every model of KB.

Corollary 3.10. Let $KB = (L, P)$ be a normal positive cq-program. If KB is satisfiable, then there is a unique model $I \subseteq HB_P$ of KB s.t. $I \subseteq J$ for all models $J \subseteq HB_P$ of KB.

Example 3.11. The cq-program (L, P) in Example 3.1 is a nondisjunctive positive cq-program with the single minimal model $\{\mathit{BadChild}(\mathit{Cain})\}$, whereas the nondisjunctive positive program (L, P') from Example 3.2 has the minimal model $\{\mathit{BadChild}(\mathit{Cain}), \mathit{BadChild}(\mathit{Romulus})\}$.

On the other hand, if a cq-program contains disjunction, then multiple minimal models of KB may exist.

Example 3.12. Consider the region program in Example 3.7. If we remove "not" from P by replacing rule (1) with

$$\begin{aligned} \mathit{visit}(L) \vee \neg\mathit{visit}(L) \leftarrow\ & \mathrm{DL}[\mathit{WhiteWine}](W), \mathrm{DL}[\mathit{RedWine}](R), \\ & \mathrm{DL}[\mathit{locatedIn}](W, L), \mathrm{DL}[\mathit{locatedIn}](R, L), \end{aligned}$$

we get a positive cq-program which has nine minimal models. The following minimal models are abbreviated versions of these models: 1. $\{\mathit{visit}(\mathit{EdnaValleyRegion}), \ldots\}$, 2. $\{\mathit{visit}(\mathit{SonomaRegion}), \ldots\}$, 3. $\{\mathit{visit}(\mathit{NapaRegion}), \ldots\}$, 4. $\{\mathit{visit}(\mathit{NewZealandRegion}), \ldots\}$, 5. $\{\mathit{visit}(\mathit{SouthAustraliaRegion}), \ldots\}$, 6. $\{\mathit{visit}(\mathit{AustralianRegion}), \ldots\}$, 7. $\{\mathit{visit}(\mathit{SantaBarbaraRegion}), \ldots\}$, 8. $\{\mathit{visit}(\mathit{CaliforniaRegion}), \ldots\}$, and 9. $\{\mathit{visit}(\mathit{USRegion}), \ldots\}$.

3.3.2 Strong answer-set semantics for cq-programs

We now define the *strong answer-set semantics* of general cq-programs. It reduces to the minimal model semantics for positive cq-programs, using a generalized transformation that removes all NAF-literals and every nonmonotonic dl-atom.

We can accommodate this with possibly nonmonotonic dl-atoms by treating them similarly as NAF-literals. This is particularly useful, if we do not know a priori whether some dl-atoms are monotonic, and determining this might be costly; notice, however, that absence of $\Cap$ in an input list of a dl-atom is a simple syntactic criterion that implies monotonicity of a dl-atom.

For any cq-program $KB = (L, P)$, we denote by DL_P the set of all ground dl-atoms that occur in $ground(P)$. We assume that KB has an associated set $DL_P^+ \subseteq DL_P$ of ground dl-atoms which are known to be monotonic, and we denote by $DL_P^? = DL_P \setminus DL_P^+$ the set of all other ground dl-atoms. An input literal of $a \in DL_P$ is a ground literal with an input predicate of a and constant symbols in Φ.

Definition 3.1. The strong dl-reduct of P relative to L and an interpretation $I \subseteq HB_P$, denoted sP_L^I, is the set of all cq-rules obtained from $ground(P)$ by

(i) deleting every cq-rule r such that either $I \models_L a$ for some $a \in B^+(r) \cap DL_P^?$, or $I \models_L l$ for some $l \in B^-(r)$; and

(ii) deleting from each remaining cq-rule r all literals in $B^-(r) \cup (B^+(r) \cap DL_P^?)$.

Notice that (L, sP_L^I) has only monotonic dl-atoms and no NAF-literals anymore. Thus, (L, sP_L^I) is a positive cq-program, and by Corollary 3.10, has a minimal model, if it is satisfiable and normal. We thus define the strong answer-set semantics of general cq-programs by reduction to the minimal model semantics of positive cq-programs as follows.

Definition 3.2. Let $KB = (L, P)$ be a cq-program. A *strong answer set* of KB is an interpretation $I \subseteq HB_P$ such that I is a minimal model of (L, sP_L^I).

Example 3.13. The minimal models shown in Example 3.11 are strong answer sets of the resp. cq-programs.

The region program KB from Example 3.7 has the following three answer sets (only the positive facts of predicates *delicate_region* and *visit* are listed, which are abbreviated by *dr* resp. *v*): 1. $\{dr(\mathit{LaneTannerPinotNoir}), dr(\mathit{WhitehallLanePrimavera}), v(\mathit{USRegion}), \ldots\}$, 2. $\{dr(\mathit{MountadamRiesling}), v(\mathit{AustralianRegion}), \ldots\}$, and 3. $\{dr(\mathit{StonleighSauvignon\text{-}Blanc}), v(\mathit{NewZealandRegion}), \ldots\}$.

The following result shows that the strong answer-set semantics of a cq-program $KB = (L, P)$ without dl-atoms coincides with the ordinary answer set semantics of P.

Theorem 3.14. Let $KB = (L, P)$ be a cq-program without dl-atoms. Then, $I \subseteq HB_P$ is a strong answer set of KB iff it is an answer set of the ordinary program P.

Proof. Let $I \subseteq HB_P$. Then, P has no dl-atoms implies $sP_L^I = P^I$. Hence, I is minimal model of (L, sP_L^I) iff I is a minimal model of P^I. Therefore, I is a strong answer set of (L, P) iff I is an answer set of P. □

The next result shows that, as desired, strong answer sets of a cq-program KB are models of KB, too, and moreover minimal models of KB if all dl-atoms are monotonic (and known as such, i.e., $DL_P^? = \emptyset$).

Theorem 3.15. Let $KB = (L, P)$ be a cq-program, and let M be a strong answer set of KB. Then, (a) M is a model of KB, and (b) M is a minimal model of KB if $DL_P = DL_P^+$.

Proof. (a) Let I be a strong answer set of KB. To show that I is also a model of KB, we have to show that $I \models_L r$ for all $r \in \mathit{ground}(P)$. Consider any $r \in \mathit{ground}(P)$. Suppose that $I \models_L l$ for all $l \in B^+(r)$ and $I \not\models_L l$ for all $l \in B^-(r)$. Then, the cq-rule r' that is obtained from r by removing all the literals in $B^-(r) \cup (B^+(r) \cap DL_P^?)$ is contained in sP_L^I. Since I is a minimal model of (L, sP_L^I) and thus in particular a model of (L, sP_L^I), it follows that I is a model of r'. Since $I \models_L l$ for all $l \in B^+(r')$ and $I \not\models_L l$ for all $l \in B^-(r') = \emptyset$, it follows that $I \models_L H(r) = H(r')$. This shows that $I \models_L r$. Hence, I is a model of KB.

(b) By part (a), every strong answer set I of KB is a model of KB. Assume that every dl-atom of KB is monotonic, that is, $DL_P = DL_P^+$. We show now that I is a minimal model of KB. Towards a contradiction, suppose the contrary, that is, there is a $J \subset I$ such that J is a model of KB. Since J is a model of KB, we obtain that J is a model of (L, sP_L^J). Since every dl-atom $a \in DL_P$ is monotonic relative to KB, it follows that $sP_L^I \subseteq sP_L^J$. Hence, J is also a model of (L, sP_L^I). But this contradicts that I is a minimal model of (L, sP_L^I). Therefore, I is a minimal model of KB. $\square$

The semantics for cq-programs without $\Cap$ can be equivalently defined in terms of HEX-programs (see [Eiter et al., 2005b] for an overview). This partial equivalence will be dealt with in full detail in Chapter 4 and is the basis for our prototype implementation of cq-programs, the dl-plugin for dlvhex. As the above results show, many of the properties of dl-programs are naturally inherited to cq-programs, like the existence of unique answer set in absence of $\Cap$ and "not," or if "not" is used in a stratified way.

Example 3.1 and 3.2 show that cq-programs are more expressive than dl-programs in [Eiter et al., 2004b, 2006a]. Furthermore, answer-set existence for KB and reasoning from the answer sets of KB is decidable if (U)CQ-answering on L is decidable, which is feasible for quite expressive DLs including $\mathcal{SHIQ}$, $\mathcal{SHOQ}$, and fragments of $\mathcal{SHOIN}$, cf. [Ortiz de la Fuente et al., 2006b,a, Glimm et al., 2007a,b]. Rosati's well-known $\mathcal{DL}$+*log* formalism [Rosati, 2006a,b], and the more expressive hybrid MKNF knowledge bases [Motik et al., 2006, Motik and Rosati, 2007] are closest in spirit to dl- and cq-programs, since they support nonmonotonic negation and use constructions from non-monotonic logics. However, their expressiveness seems to be different from dl- and cq-programs. It is reported in [Motik et al., 2006] that dl-programs (and hence cq-programs) can not be captured using MKNF rules. In turn, the semantics of $\mathcal{DL}$+*log*-programs inherently involves deciding containment of CQs in UCQs, which seems to be inexpressible in cq-programs.

In the remainder of this thesis, however, we focus on implementation details in Chapter 4 and equivalence preserving rewritings of (u)cq-atoms, which can be exploited for program optimization as shown in Chapter 5.

Yes, I programmed it in for you. Four million lines of BASIC!

—Kif in Futurama, *Kif Gets Knocked Up A Notch*

4 Implementation

Within this chapter, we present the dl-plugin, a library written in C++ (cf. [Stroustrup, 1997]) for the HEX-program solver dlvhex. dlvhex has a builtin plugin mechanism, such that writing code for external atoms with program optimization highlights can be quickly accomplished. For this purpose, we introduce a partial equivalence between HEX-programs and cq-programs, see Section 2.6 and 3, respectively. This will be our second result.

As shown in Section 1.2, the Semantic Web architecture is composed of several layers. dlvhex' plugins currently provide an implementation for querying the RDF Layer by virtue of the RDF- and the SPARQL-plugin. To complete its support for the Semantic Web, dlvhex needs support for ontologies. In our approach, the rule layer of dlvhex sits on top of ontologies, and due to the input mechanism of external atoms the ontology layer sits on top of the rule component, thus arranging dlvhex to a very versatile and comfortable Semantic Web programming environment; as a matter of fact, the dl-plugin contributes to the underpinning of this full-fledged Semantic Web reasoning architecture.

The dl-plugin supports various external atoms for querying DL concepts and roles, issuing conjunctive and unions of conjunctive queries to DL-KBs, and provides the means for optimizing dl- and cq-programs as well as transforming dl- and cq-programs into HEX syntax.

In the following text, a principal architectural overview of the dl-plugin is shown. Moreover, the interface from the dl-plugin to the RacerPro reasoner and from dl-programs to HEX-programs will be given in full detail. The last section provides usage information of the dl-plugin in HEX-program.

4.1 Architectural Overview

The dlvhex plugin architecture is roughly divided in three portions, which may depend on each other: (i) an *Initialization*, (ii) a *Program Rewriting*, and (iii) an *External Atoms* part. In part (i), dlvhex initializes the plugin and gathers information relevant to the plugin's external atoms, optimization features, etc. Just before dlvhex will try to parse the input program, it calls the converter of each plugin, in order to transform the special syntax or syntactic sugar of a program specifically designed for a particular plugin into proper HEX-syntax. These conversion and the optimization procedures of programs belong to part (ii) of dlvhex' plugin architecture. Finally, during program evaluation, dlvhex may ask queries to external atoms defined in the plugin. All the external atoms a plugin has

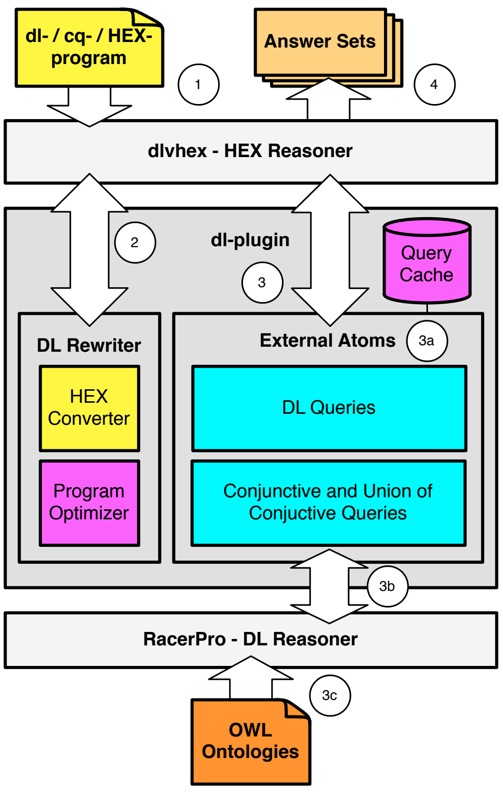

Figure 4.1: Overview dl-plugin

implemented belong to part (iii)—an alternative viewpoint is to see this part as a query evaluation unit. For a full description of the plugin mechanism and architecture, we refer to the doctoral thesis [Schindlauer, 2006]; annotated code and the corresponding class commentary is part of the online documentation of the dlvhex project.[1]

The principle building blocks of the dl-plugin are depicted in Figure 4.1. Leaving out the initialization phase, we can see that the program rewriting part is made up of the *HEX Converter* and the *Program Optimizer*; the latter optimizes a HEX-program by syntactic transformations. The *External Atoms* part comprises the *DL Queries* unit—like the external atoms *dlC*, *dlR*, and *dlDR*—, while the *Conjunctive and Union of Conjunctive Queries* are dealt with in the external atoms *dlCQ*, *dlUCQ*. On top of that, the *Query Cache* accelerates query processing; it stores answers from previously processed queries. In the following sections, we will shed light on each component and explain the implementation details. The interaction between dlvhex and dl-plugin is shown in the next sequential steps of Figure 4.1:

(1) Input: dl-, cq-, or HEX-program;

(2) convert dl- and cq-programs to HEX-syntax, and optimize by program rewriting;

(3) repetitive querying: (3a) query dl-plugin (check cache), (3b) transform query to RacerPro query, and (3c) retrieve answer;

(4) output: answer sets of input program.

As it is evident from this listing, dl-plugin is the bridge linking dlvhex and RacerPro. Multiple external atom queries may occur during the evaluation of a given logic program,

[1]`http://www.kr.tuwien.ac.at/research/dlvhex/`

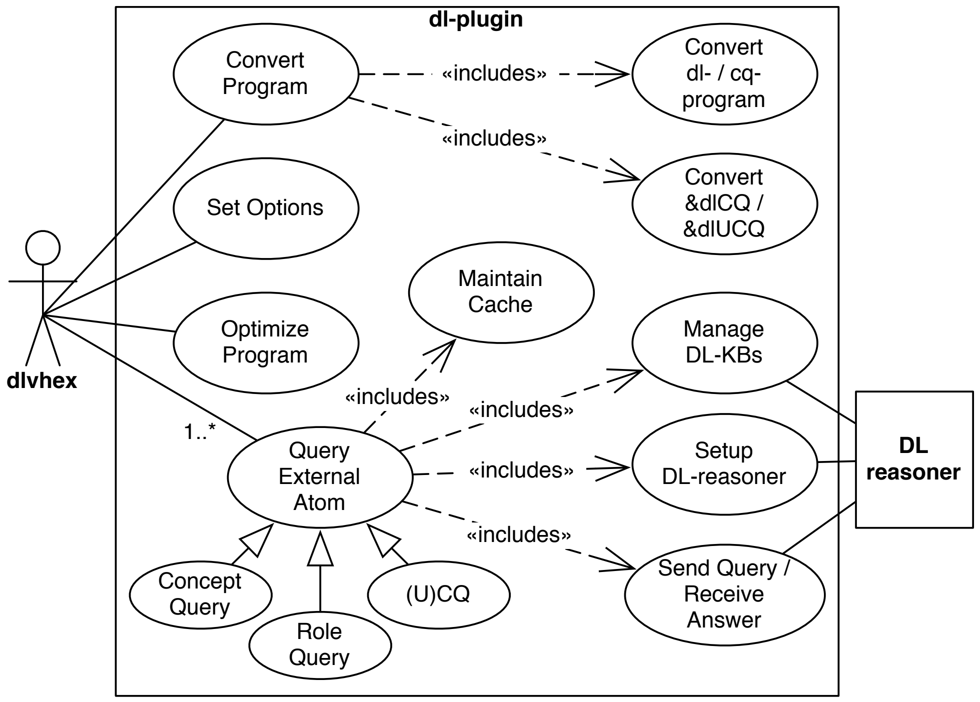

Figure 4.2: Use Case Diagram dl-plugin

hence optimization is mandatory to minimize the effects of querying external components.

4.1.1 Plugin use cases

The use cases for the dl-plugin are shown in the use case diagram in Figure 4.2. This provides a first overview of dl-plugin's usage scenarios. The primary actor in our setting is dlvhex, which actuates the reasoning machinery. The DL-reasoner—RacerPro in our implementation—plays the role of a supporting actor; it waits for requests from the dl-plugin and answers queries. dlvhex on the other hand is always proactive. The dl-plugin offers four main services for dlvhex: (i) "Set Options," (ii) "Convert Program," (iii) "Optimize Program," and (iv) "Query External Atom." In use case (i), dlvhex may set miscellaneous options in the dl-plugin. Usually, options are provided as a list of command line arguments of the dlvhex program. Use case (ii) takes an input program and converts it according to the translation given in Section 4.4. In use case (iii), dl-plugin may optimize a program by means of the procedures described in Section 4.5. Finally, use case (iv) shows the requirements for the external atoms provided by the dl-plugin.

4.1.2 Plugin components

In this section, we will review the pieces of the dl-plugin and its dependencies. To this end, we use component diagrams by the Unified Modeling Language (UML) [Rumbaugh et al., 2004, 2005].

First of all, the overall component overview of the dl-plugin is displayed in Figure 4.3; the three main constituents are dlvhex, dl-plugin, and the RacerPro DL-reasoner. dlvhex uses the interfaces `PluginInterface`, `PluginAtom`, `Options`, `PluginConverter`, and `Plugin-Optimizer`, while dl-plugin implements some of the interfaces defined in the plugin API of dlvhex in the components `ExtAtom`, `DLConverter`, and `DLOptimizer`. Further components

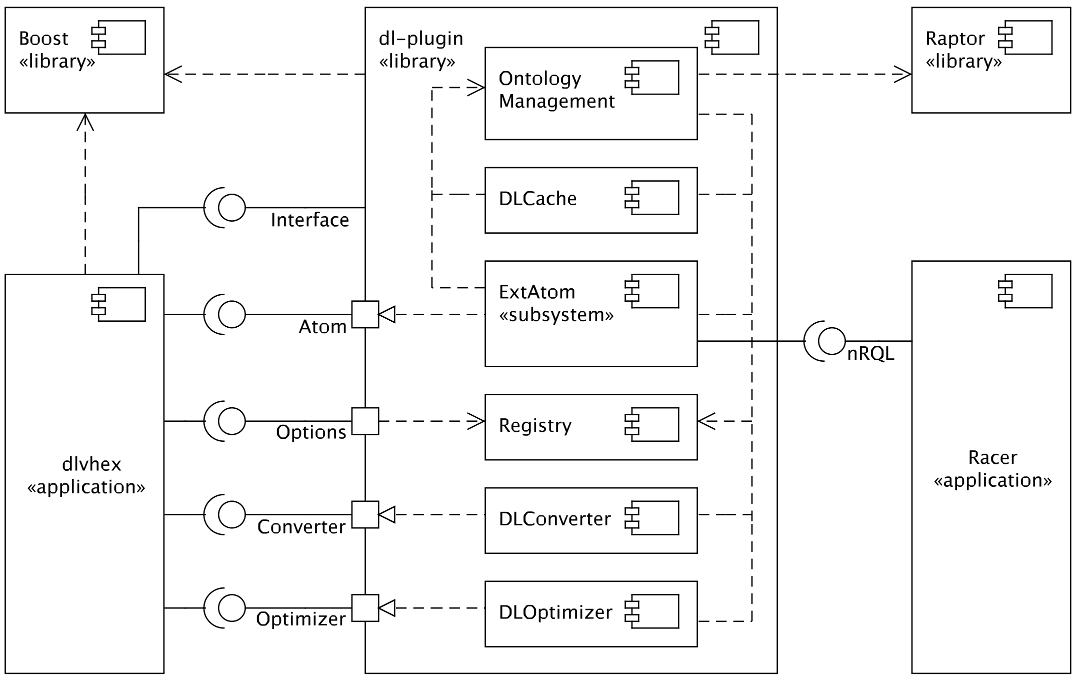

Figure 4.3: Component Diagram dl-plugin

of the dl-plugin are the `Registry`, which provides information for the `Options` interface and all the other parts implemented in the plugin, the `OntologyManagement`, which holds information concerning locations and structures of OWL ontologies, and the `DLCache` module for caching simple dl-queries.

Moreover, dl-plugin uses external libraries for rapid prototyping and RDF support. To aid our rapid prototyping needs, Boost[2] provides easy accessible and high quality C++ libraries used throughout the dl-plugin. The Raptor[3] RDF parsing library—which is part of the more general Redland RDF framework—appropriates parsers for the many RDF syntax variants, hence accessing OWL ontologies on a strict syntactic level comes for free. Future work may include support for the whole Redland[4] framework, since support for triple-stores (cf. [Beckett and Grant, 2003]) may provide convenient and fast access to relational databases for storing RDF data in highly tuned RDBMS. This is in line with the property of polynomial-time reasoning in lightweight DLs such as the DL-Lite [Calvanese et al., 2005] and $\mathcal{EL}$ [Baader et al., 2005a] family of DLs, since they are tailored for an easy translation into relational structures and provide fast reasoning algorithms. On the other hand, the SPARQL query language, as implemented in the Rasqal library,[5] supplies the appliances for accessing other DL-reasoners such as KAON2 and Pellet with built-in support for SPARQL over DLs (cf. [Sirin and Parsia, 2007]).

The dl-plugin supports the program optimization procedures described in the upcoming Chapter 5. In particular, all Algorithms 3–5 are implemented in the `DLOptimizer` unit.

A more detailed view of the internal components from the `ExtAtom` subsystem is shown in Figure 4.4. Here, we can see the cutting line between the simple concept and role querying atoms &*dlC*, &*dlR*, and &*dlDR*, and the more complex CQ and UCQ external atoms. The former atoms may use `DLCache`, while the latter do not have any support

[2]`http://www.boost.org/`
[3]`http://www.librdf.org/raptor/`
[4]`http://www.librdf.org/`
[5]`http://www.librdf.org/rasqal/`

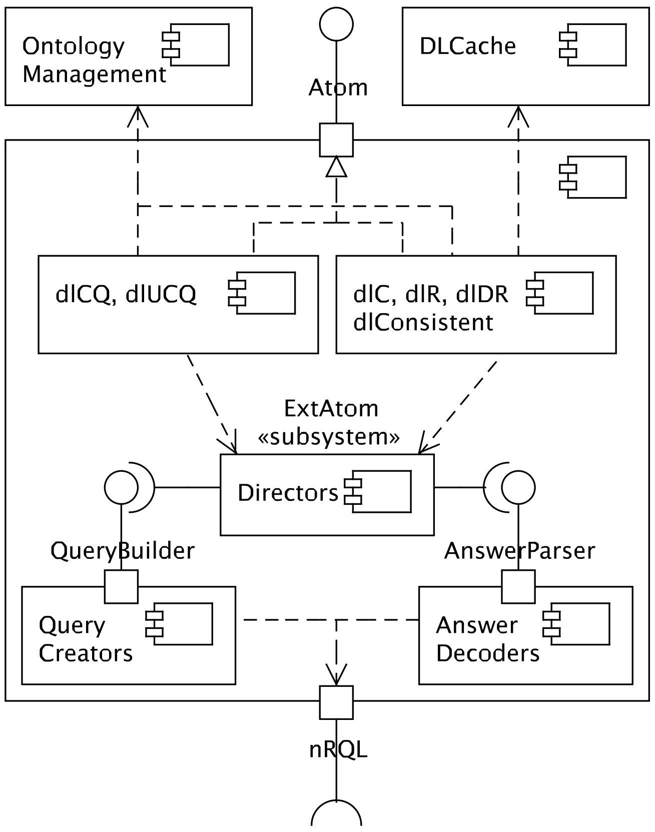

Figure 4.4: Component Diagram External Atom Subsystem

for caching queries right now (see the discussion in Section 4.5.2). Both pieces use the `Directors` component, which delegates query creation and answer parsing by means of `QueryBuilder` and `AnswerParser`, which are in direct contact with the `nRQL` interface of RacerPro, a query language for DL-KBs in RacerPro. This also provides hints on how to adapt the dl-plugin for accessing different description logics reasoners. In principal, only `QueryBuilder` and `AnswerParser` must be customized for future accommodations.

4.2 Querying DL-KBs in HEX-Programs

As shown in [Eiter et al., 2005b], the HEX-program formalism is ideal for modelling ASP extensions by means of external atoms. cq-programs are such an extension, although they do not come in HEX-syntax. In Section 4.4, we introduce a partial equivalence from dl- and cq-programs to HEX-programs. Hence, we only need to implement external atoms in order to implement dl- and cq-programs. Moreover, the DL external atoms provided in our implementation have the ability to specify the DL-KB name as input parameter, thus we are, in principle, able to integrate multiple ontologies with HEX-programs.

The dl-plugin implements several external atoms for querying description logics knowledge bases. In the following section, we will define the syntax and the semantics of these DL external atoms.

Definition 4.1. Let u be a constant string representing the URI of a description logic knowledge base, $a, b, c, d \in \mathcal{P}$ be predicate names, and q be a constant string, such that $q(X_1, \ldots, X_m)$ is a dl-, cq- or ucq-query for an output length m of the respective external atom. A description logic knowledge base querying external atom, or short *DL external atom*, is of form

$$\begin{aligned}
\&dlC[u,a,b,c,d,q](X_1) &\text{ for dl-query } q(X_1),\\
\&dlR[u,a,b,c,d,q](X_1,X_2) &\text{ for dl-query } q(X_1,X_2),\\
\&dlDR[u,a,b,c,d,q](X_1,X_2) &\text{ for datatype dl-query } q(X_1,X_2),\\
\&dlCQ_m[u,a,b,c,d,q](X_1,\dots,X_m) &\text{ for cq-query } q(X_1,\dots,X_m), \text{ and}\\
\&dlUCQ_m[u,a,b,c,d,q](X_1,\dots,X_m) &\text{ for ucq-query } q(X_1,\dots,X_m)
\end{aligned}$$

such that each DL external atom has type signature $(\mathsf{c},\mathsf{p},\mathsf{p},\mathsf{p},\mathsf{p},\mathsf{c})$. For an interpretation I, the associated oracle functions for the above-stated external atoms are then

$$\begin{aligned}
&f_{\&dlC}(I,u,a,b,c,d,q,X_1),\\
&f_{\&dlR}(I,u,a,b,c,d,q,X_1,X_2),\\
&f_{\&dlDR}(I,u,a,b,c,d,q,X_1,X_2),\\
&f_{\&dlCQ_m}(I,u,a,b,c,d,q,X_1,\dots,X_m), \text{ and}\\
&f_{\&dlUCQ_m}(I,u,a,b,c,d,q,X_1,\dots,X_m),
\end{aligned}$$

respectively.

Note that $\&dlCQ_m$ and $\&dlUCQ_m$ have a specific output arity m, so it is not possible to capture all possible arities $m \geq 0$ of (U)CQs in the plugin. Hence, we set the maximal output arity of these external atoms to a predefined constant k and define $\&dlCQ_m$ and $\&dlUCQ_m$ with m ranging over $0 \leq m \leq k$. This stems from the observation that dlvhex expects all known external atoms to be present at initialization time, i.e., before the dl-plugin knows about the given program. The dl-plugin describes output list members by a bit field, hence k is set to `sizeof(int)` (32 on our computers). Note that this limitation could be overcome by creating $\&dlCQ_i$ and $\&dlUCQ_i$ external atoms on demand at the transformation step in dlvhex. In the sequel, we omit m and write $\&dlCQ$ and $\&dlUCQ$ when its clear from the context. Indeed, the dl-plugin supports this syntax and rewrites every occurrence of $\&dlCQ$ and $\&dlUCQ$ to the respective external atom.

Definition 4.2. Let P and N be lists of predicate symbols called *positive* and *negative filter predicate list*, respectively. For an interpretation I, let $P(I)$ be the *positive projection* of I on predicates from P, then build fresh positive literals by removing the predicate names and using the first arguments as new predicate names; $N(I)$ is the *negative projection*, which removes the predicate names and builds new negative literals with first arguments as predicate symbol.

The next example shows the outcome of $P(I)$ and $N(I)$ for a given interpretation I.

Example 4.1. Let $I = \{p(a,b), q(c,d,e)\}$ be an interpretation. Then, for $P = p$ and $N = q$, $P(I) = \{a(b)\}$ and $N(I) = \{\neg c(d,e)\}$.

We are now ready to define the oracle functions semantics for $f_{\&dlC}$, $f_{\&dlR}$, $f_{\&dlDR}$, $f_{\&dlCQ_m}$, and $f_{\&dlUCQ_m}$. When it is clear from the context, we omit the URL constant string u for a DL-KB L and simply write L.

Definition 4.3. Let $c_1,\dots,c_m$ be constant symbols from $\mathcal{C}$, $P = a,c$ and $N = b,d$ be positive and negative filter predicate lists, u and L be a URI and a description logics knowledge base for u, respectively, and let $ans(q(\vec{X}), L)$ be the set of answers of $q(\vec{X})$ on L. We define

$$
\begin{aligned}
&f_{\&dlC}(I, u, a, b, c, d, q, c_1) = 1 \text{ iff } L \cup P(I) \cup N(I) \models q(c_1),\\
&f_{\&dlR}(I, u, a, b, c, d, q, c_1, c_2) = 1 \text{ iff } L \cup P(I) \cup N(I) \models q(c_1, c_2),\\
&f_{\&dlDR}(I, u, a, b, c, d, q, c_1, c_2) = 1 \text{ iff } L \cup P(I) \cup N(I) \models q(c_1, c_2),\\
&f_{\&dlCQ_m}(I, u, a, b, c, d, q, c_1, \dots, c_m) = 1 \text{ iff } \langle c_1, \dots, c_m \rangle \in\\
&\qquad\qquad\qquad\qquad\qquad ans(q(X_1, \dots, X_m), L \cup P(I) \cup N(I)),\\
&\text{and } f_{\&dlUCQ_m}(I, u, a, b, c, d, q, c_1, \dots, c_m) = 1 \text{ iff } \langle c_1, \dots, c_m \rangle \in\\
&\qquad\qquad\qquad\qquad\qquad ans(q(X_1, \dots, X_m), L \cup P(I) \cup N(I)).
\end{aligned}
$$

Now we are capable of implementing each oracle function in a DL-reasoner. The dl-plugin uses RacerPro as host reasoner, but other DL-reasoners could be used as well. For this, we will outline the RacerPro query syntax of each DL query type with the associated external atom in the next section.

Figure 4.5 illustrates the general workflow for processing a query in a DL external atom of the dl-plugin. The diagram shows that specific optimization methods take place during the evaluation of a query. More details on the optimization methods will be in Section 4.5 and Chapter 5.

4.3 Interfacing RacerPro with the dl-Plugin

RacerPro is a description logics reasoner with support for expressive ABox queries like concept and role, conjunctive, and union of conjunctive queries over ABoxes. Moreover, it provides DL knowledge base management commands to simultaneously access multiple ontologies.

The RacerPro system comes in form of a server binary, which awaits commands from client programs through a TCP/IP connection. The system manuals [Rac, 2005a,b] give more detail on this TCP API, the command line options, query and answer syntax, as well as configuration commands.

4.3.1 New Racer Query Language (nRQL)

The RacerPro system had only limited support for ABox queries before version 1.8. In particular, only concept, role, and role-fillers instance retrieval were supported. To overcome this limitation, nRQL had been introduced in RacerPro 1.8 and was since then RacerPro's standard query language for issuing more expressive queries to a DL-KB. In our setting, we use RacerPro 1.9.

nRQL is capable of querying ABoxes (and TBoxes) in a very expressive manner, i.e., any monotonic combination of conjunctive and disjunctive query expressions over DL-KBs is supported—a limited form of nonmonotonic queries is expressible in nRQL too.

In the following, we give a brief introduction into nRQL and how this query language is used in dl-plugin. The definitions of the syntax and semantics for nRQL is given in full detail in [Wessel and Möller, 2006] and in the RacerPro system manuals [Rac, 2005a,b]. We just remark that RacerPro supports only limited, i.e., safe CQs and UCQs, where existential variables are grounded with respect to the universe (cf. the discussion in Section 2.8). For our purposes, we do not need the full range of nRQL's possibilities, e.g., we do not use the tuple-at-a-time feature or use named query processing, although in future work, such

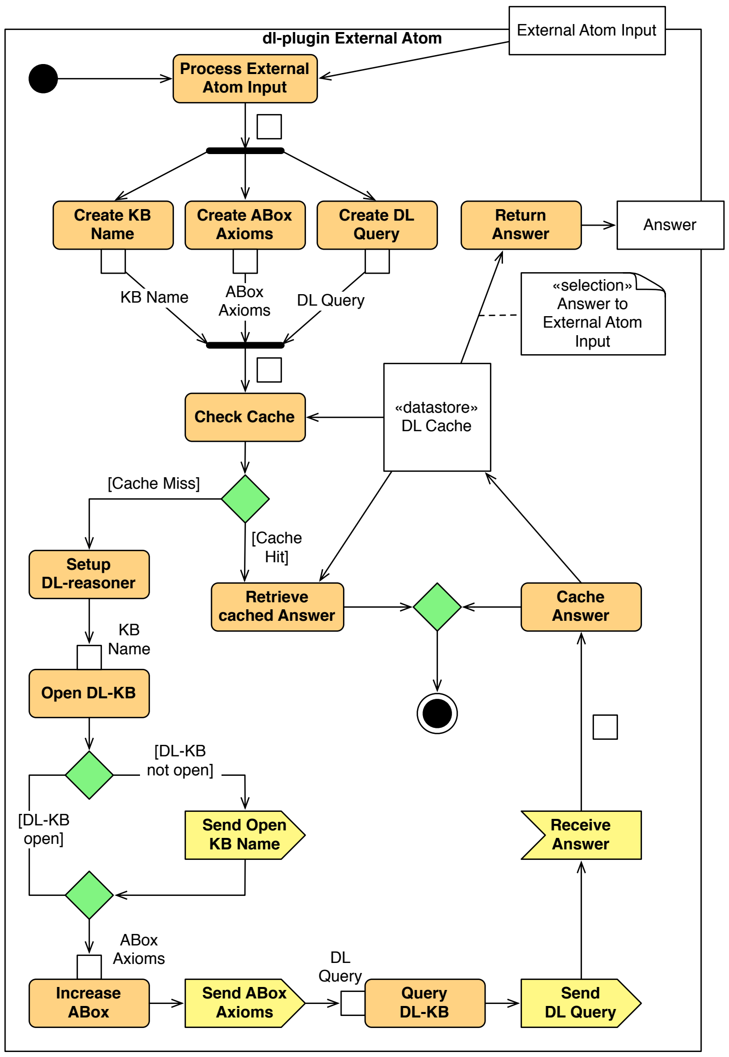

Figure 4.5: Activity Diagram External Atom Query Processing

enhanced query management facilities might be useful for implementation shortcuts. We now list the used features of nRQL in regard to a querying session. See Figure 4.5 for the states that may arise during DL external atom query processing.

4.3.2 Augmenting the DL knowledge base

In the "Send ABox Axioms" and "Create ABox Axioms" states, we have to create ABox Concept and Role assertions with the operator `instance` and `related`. The syntax for the operation of adding an individual to a concept is

```
(instance IN C)
```

where `IN` is an individual and `C` is a concept expression. Adding a pair of individuals to a role is performed by

```
(related IN1 IN2 R)
```

where `IN1` and `IN2` are individuals and `R` is a role expression.

RacerPro supports DL-KB ABox enhancement by means of the `state` command. The `state` command expects a list of `instance` and `related` commands and increases the current DL-KB by these ABox assertions.

```
(state (instance A B) ...  (related C D E))
```

The next query will then be to the extended ABox.

Similarly, nRQL queries provide support for augmenting the current ABox, but direct in the query statement:

```
(retrieve-under-premise ((instance A B)···(related C D E)) ··· ).
```

After this `retrieve-under-premise` query, the ABox will be in the same state as before the query.

Using the former operations, we can implement $L \cup P(I) \cup N(I)$ in a straightforward way by creating a list of `instance` and `related` statements. With these lists at hand, we are in a position to decide whether to invoke `state` or `retrieve-under-premise`, depending on the query type.

Note that the negative role assertion $\neg R(b, c)$—we need this when roles are involved in the $\uplus$ operator in cq-programs or the fifth parameter in the input list of a DL external atom is a known predicate—is not directly expressible in $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, instead, an equisatisfiable expression is needed to resolve this issue. One solution is to use the next theorem from [Lukasiewicz, 2007]:

Theorem 4.2 ([Lukasiewicz, 2007]). Let L be a description logic knowledge base, and let $R(b, c)$ be a role membership axiom. Then, $L \cup \{\neg R(b, c)\}$ is satisfiable iff $L \cup \{B(b), C(c), \exists R.C \sqsubseteq \neg B\}$ is satisfiable, where B and C are two fresh atomic concepts.

Other solutions exist, like $L \cup \{\neg R(b, c)\}$ is satisfiable iff $L \cup \{\neg(\exists R.\{b\})(c)\}$ is satisfiable (see [Eiter et al., 2004c]), but using this assertion involves nominals, for which current DL-reasoners have no full support yet. In fact, RacerPro supports only approximated nominals (cf. [Rac, 2005b]). On the other hand, Theorem 4.2 includes a TBox axiom, which implies that we have to augment the TBox and the ABox of the DL-KB.

As an aside, the upcoming OWL 1.1 standard will support negative property membership assertions.[6]

4.3.3 DL-KB management

Before we issue a query to RacerPro, we have to make sure that we select the correct DL-KB. To create a pristine knowledge base, we use `(owl-read-document URI)` or `(owl-read-file FILE)`, which receives the URI or reads the file pointed to by `URI` and `FILE`, resp. More commands exist for receiving the list of opened DL-KBs, or removing DL-KBs; see [Rac, 2005a,b] for a detailed account.

4.3.4 Concept queries

Using the *&dlC* external atom, we are free to ask two different forms of concept queries to a DL knowledge base: *instance retrieval* and *instance checking* queries.

Instance retrieval queries are queries of form

```
(concept-instances C),
```

[6]`http://www.w3.org/Submission/2006/SUBM-owl11-overview-20061219/#2.1`

where `C` is a concept expression. This operation returns all individuals belonging to the specified concept expression `C`, so it is well suited for non-ground *&dlC* external atoms.

Ground *&dlC* external atoms are implemented using the instance checking query

```
(individual-instance?  IN C).
```

This query decides whether the individual `IN` is a member of concept `C`.

4.3.5 Role queries

Role query answering is the domain of the *&dlR* and *&dlDR* external atoms. In case of *&dlR*, we can differentiate between pure non-ground role retrieval queries (i.e., ask for all pairs of a given role expression), individual fillers for a given role and an individual, and ground pair checking. The datatype role querying external atom *&dlDR* supports similar queries, but on a more restricted basis. We begin with the *&dlR* external atom.

In the pure non-ground situation, that is, both output terms of a *&dlR* external atom are variables, we use the expression

```
(related-individuals R)
```

to query a role `R` and retrieve all related pairs.

A ground *&dlR* atom uses the query

```
(individuals-related?  i1 i2 R)
```

to check whether the pair (`i1`,`i2`) belongs to `R`.

For non-ground *&dlR* external atoms with exactly one constant output term, we use the query

```
(individual-fillers i R)
```

for an external atom $\&dlR[L, a, b, c, d, \text{“}R\text{”}](i, X)$, and we translate the external atom $\&dlR[L, a, b, c, d, \text{“}R\text{”}](X, i)$ to the query

```
(individual-fillers i (inv R)).
```

Note that `(inv R)` is the inverse role `R`$^{-}$.

Datatype role queries needs special care. RacerPro supports OWL datatype properties, but querying such roles is only supported in nRQL queries (more on such queries in the next section). In addition to this characteristic, before we query datatype roles, we have to enable RacerPro's datatype role capabilities by the command `(ENABLE-DATA-SUBSTRATE-MIRRORING)`. After that, datatype role querying is enabled.

The non-ground external atom $\&dlDR[L, a, b, c, d, \text{“}R\text{”}](X, Y)$ uses the nRQL expression

```
(retrieve ($?X $?Y) ($?X $?Y (:owl-datatype-role R)))
```

to retrieve all pairs of the datatype property `R`. A similar expression is used for datatype role filler queries in $\&dlDR[L, a, b, c, d, \text{“}R\text{”}](i, X)$:

```
(retrieve ($?X) (and ($?X $?Y (:owl-datatype-role R)) (same-as $?X i))).
```

Note that the `same-as` equality atom compares its parameters on a pure syntactic basis, i.e., this equality does not have first-order semantics. A final remark on the other query types: we cannot ask pure ground datatype role queries or use the “inverse” datatype role querying pattern in *&dlDR* external atoms, for instance $\&dlDR[L, a, b, c, d, \text{“}R\text{”}](X, \text{“}2.5\text{”})$, as both query forms are not supported in RacerPro.

4.3.6 Conjunctive and union of conjunctive queries

Conjunctive and union of conjunctive queries are expressed using the nRQL query constructors `retrieve` and `retrieve-under-premise`. The first one just queries the given ABox, while the second one increases the ABox by a list of ABox assertions right before the querying takes place. Thus, `retrieve-under-premise` comes quite handy when we have to augment the ABox before the querying part, i.e., it is a neat way to implement $P(I) \cup N(I) \neq \emptyset$.

nRQL is equipped with the `and` and `union` query constructors. The two of them are used to build complex (U)CQs from simple concept query atoms of form

```
($?X C)
```

and from role query atoms of form

```
($?X $?Y R),
```

where `$?X` and `$?Y` are variables—instead of variables, we use named individuals for ground or filler queries—, and `C` and `R` is a concept and role expression, respectively. A conjunctive query expression is

```
(and Q1 ... Qn),
```

where each `Qi` is a simple concept or role query atom. A union of conjunctive queries expression is of form

```
(union CQ1 ... CQm),
```

where each `CQi` is a conjunctive query expression.

We are now fit to express our external atoms as nRQL queries by translating *&dlCQ*'s conjunctive query to the nRQL expression

```
(retrieve ($?X1 ... $?Xn) (and Q1 ... Qn)),
```

whereas *&dlUCQ*'s union of conjunctive queries are transformed into

```
(retrieve ($?X1 ... $?Xn) (union CQ1 ... CQk)),
```

where `(and Q1 Q2 ... Qn)` and `(union CQ1 ... CQk)` are nRQL conjunctive query and union of conjunctive queries expressions, resp.

Example 4.3. In order to show some example *&dlCQ* and *&dlUCQ* external atoms, we now illustrate the corresponding nRQL queries.

The atom $\&dlCQ[L, a, b, c, d, \text{“}C(X), R(X,Y)\text{”}](X)$ issues the nRQL expression

```
(retrieve ($?X) (and ($?X C) ($?X $?Y R)))
```

while $\&dlUCQ[L, a, b, c, d, \text{“}C(X), R(Y,Z) \vee D(X), R(X,Z)\text{”}](X,Z)$ will be evaluated in the query

```
(retrieve ($?X $?Z) (union
                       (and ($?X C) ($?Y $?Z R))
                       (and ($?X D) ($?X $?Z R))
                     )
           )
```

4.4 Interfacing dl-/cq-Programs with HEX-Programs

In order to transform dl- and cq-programs into HEX-programs, we have to point out that this correspondence is only partial. This is due to the assumption that all external atoms in HEX-programs behave in a monotonic fashion, but the $\cap\!\!\!-$ operator in dl- and cq-programs is nonmonotonic, thus leading to a transformation which only supports $\uplus$ and $\cup\!\!\!-$ in its input list. Moreover, in dl-atoms, we allow certain cyclic dependencies between the input list and the rules of the program, such constructs might not be allowed by HEX-programs due to the domain-expansion safe condition.

Definition 4.4. Let (L, P) be a cq-program. For a dl-atom $a = \mathrm{DL}[\lambda, q](\vec{X})$ in P, let $\pi_L(a)$ be a set of auxiliary HEX-rules which transforms the input list $\lambda = S_1 op_1 p_1, \ldots, S_m op_m p_m$ of a w.r.t. a DL-KB L as follows:

- $\pi_L(a) = \bigcup_{i=1}^{m} \pi_L(\#\lambda, S_i op_i p_i)$, where $\#\lambda$ maps every λ to a unique number, and
- $\pi_L(n, S\, op\, p) = \begin{cases} pc_n(\text{“}S\text{”}, X) \leftarrow p(X) & op = \uplus \text{ and } S \text{ concept in } L, \\ pr_n(\text{“}S\text{”}, X, Y) \leftarrow p(X, Y) & op = \uplus \text{ and } S \text{ role in } L, \\ mc_n(\text{“}S\text{”}, X) \leftarrow p(X) & op = \cup\!\!\!- \text{ and } S \text{ concept in } L, \\ mr_n(\text{“}S\text{”}, X, Y) \leftarrow p(X, Y) & op = \cup\!\!\!- \text{ and } S \text{ role in } L. \end{cases}$

Moreover, let $\pi_L(r)$ be a set of HEX-rules, which transforms each dl-atom in a cq-rule r w.r.t. L as follows:

$$\pi_L(r) = \bigcup_{b \in B_{dl}(r)} \pi_L(b).$$

Note that all pc_n, pr_n, mc_n, mr_n are fresh predicate names not occurring in P, and that every $\pi_L(a)$ is a definite horn HEX-program with a single answer set.

Now we are ready to define τ_L as transformation of atoms, rules, and programs in dl- and cq-syntax into a HEX-program, i.e., a translation of a cq-program (L, P) into a HEX-program $\tau_L(P)$.

Definition 4.5. Let (L, P) be a cq-program. Then, $\tau_L(P)$ is a HEX-program with DL external atoms and inductively defined as

- $\tau_L(P) = \bigcup_{r \in P} \tau_L(r)$;
- for a cq-rule $r = a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \mathrm{not}\, b_{m+1}, \ldots, \mathrm{not}\, b_n$,

  $$\tau_L(r) = \{a_1 \vee \cdots \vee a_k \leftarrow \tau_L(b_1), \ldots, \tau_L(b_m), \mathrm{not}\, \tau_L(b_{m+1}), \ldots, \mathrm{not}\, \tau_L(b_n)\} \cup \pi_L(r);$$

- for a dl-atom $a = \mathrm{DL}[\lambda; q](\vec{X})$ and $n = \#\lambda$,

  $$\tau_L(a) = \&dlT[L, pc_n, mc_n, pr_n, mr_n, q](\vec{X}),$$

  where $\&dlT$ is $\&dlC$, $\&dlR$, $\&dlDR$, $\&dlCQ$, and $\&dlUCQ$ if $q(\vec{X})$ is a concept dl-query, role dl-query, datatype role dl-query, CQ, and UCQ, respectively; and

- for an ordinary atom a,

  $$\tau_L(a) = a.$$

The next example shows a simple translation from a cq-program into a corresponding HEX-program using τ_L from the earlier definition.

Example 4.4. Let (L, P) be a cq-program, where L is the DL knowledge base in Example 2.5 and P is the following set of cq-rules:

$$\begin{aligned}
isMother(X, Y) &\leftarrow mother(X), parent(X, Y).\\
isFather(X, Y) &\leftarrow father(X), parent(X, Y).\\
mother(X) &\leftarrow \mathrm{DL}[motherOf \uplus isMother; motherOf](X, Y).\\
father(X) &\leftarrow \mathrm{DL}[fatherOf \uplus isFather; fatherOf](X, Y).\\
&parent(a, b).\\
&mother(a).
\end{aligned}$$

The HEX-program $\tau_L(P)$ consists of following rules:

$$\begin{aligned}
isMother(X, Y) &\leftarrow mother(X), parent(X, Y).\\
isFather(X, Y) &\leftarrow father(X), parent(X, Y).\\
mother(X) &\leftarrow \&dlR[L, pc_0, mc_0, pr_1, mr_0, motherOf]\,(X, Y).\\
pr_1(motherOf, X, Y) &\leftarrow isMother(X, Y).\\
father(X) &\leftarrow \&dlR[L, pc_0, mc_0, pr_2, mr_0, fatherOf]\,(X, Y).\\
pr_2(fatherOf, X, Y) &\leftarrow isFather(X, Y).\\
&parent(a, b).\\
&mother(a).
\end{aligned}$$

The following example is more involved, as it contains (U)CQs and dl-atoms with long input lists.

Example 4.5. Let (L, P) be a cq-program with P as

$$\begin{aligned}
a(X) &\leftarrow \mathrm{DL}[R \uplus r; \neg D](X).\\
a(X) &\leftarrow \mathrm{DL}[C \uplus c, D \uplus d, R \cup r; R(X, Y) \vee C(X)](X), \mathrm{DL}[R \uplus r; \neg C](X).\\
r(X, Y) &\leftarrow c(X), d(Y).\\
&c(c_1).\\
&c(c_2).\\
&d(d_2).
\end{aligned}$$

and let L be the DL-KB

$$\begin{aligned}
\forall R.D &\sqsubseteq C\\
R(c_1, d_1)&
\end{aligned}$$

Then, $\tau_L(P)$ is the HEX-program

$$\begin{aligned}
a(X) &\leftarrow \&dlC[L, pc_0, mc_0, pr_1, mr_0, \text{“}\neg D\text{”}](X).\\
pr_1(\text{“}R\text{”}, X, Y) &\leftarrow r(X, Y).\\
a(X) &\leftarrow \&dlUCQ[L, pc_2, mc_0, pr_0, mr_2, \text{“}R(X, Y) \vee C(X)\text{”}](X),\\
&\qquad \&dlC[L, pc_0, mc_0, pr_1, mr_0, \text{“}\neg C\text{”}](X).\\
pc_2(\text{“}C\text{”}, X) &\leftarrow c(X).\\
pc_2(\text{“}D\text{”}, X) &\leftarrow d(X).\\
mr_2(\text{“}R\text{”}, X, Y) &\leftarrow r(X, Y).\\
r(X, Y) &\leftarrow c(X), d(Y).\\
&c(c_1).\\
&c(c_2).\\
&d(d_2).
\end{aligned}$$

Note that the τ_L transformation still cannot capture all dl- and cq-programs. The next example explains why.

Example 4.6. Consider dl-program (L_N, P_N) from Example 2.7. The translated HEX-program $\tau_{L_N}(P_N)$ comprises following rules:

$$\begin{aligned}
&\mathit{newnode}(\mathit{add_1}).\\
&\mathit{newnode}(\mathit{add_2}).\\
\mathit{overloaded}(X) &\leftarrow \&\mathit{dlC}[L_N, pc_0, mc_0, pr_1, mr_0, \text{“}\mathit{HighTrafficNode}\text{”}](X).\\
pr_1(\text{“}\mathit{wired}\text{”}, X, Y) &\leftarrow \mathit{connect}(X, Y).\\
\mathit{connect}(X, Y) &\leftarrow \mathit{newnode}(X), \&\mathit{dlC}[L_N, pc_0, mc_0, pr_0, mr_0, \text{“}\mathit{Node}\text{”}](X),\\
&\qquad \text{not } \mathit{overloaded}(Y), \text{not } \mathit{excl}(X, Y).\\
\mathit{excl}(X, Y) &\leftarrow \mathit{connect}(X, Z), \&\mathit{dlC}[L_N, pc_0, mc_0, pr_0, mr_0, \text{“}\mathit{Node}\text{”}](Y),\\
&\qquad Y \neq Z.\\
\mathit{excl}(X, Y) &\leftarrow \mathit{connect}(Z, Y), \mathit{newnode}(Z), \mathit{newnode}(X), Z \neq X.\\
&\mathit{excl}(\mathit{add_1}, n_4).
\end{aligned}$$

This program is not domain-expansion safe, since

$$\mathit{overloaded}(X) \leftarrow \&\mathit{dlC}[L_N, pc_0, mc_0, pr_1, mr_0, \text{“}\mathit{HighTrafficNode}\text{”}](X)$$

is not strongly safe, because *overloaded* does not strictly depend on pr_1 from

$$\&\mathit{dlC}[L_N, pc_0, mc_0, pr_1, mr_0, \text{“}\mathit{HighTrafficNode}\text{”}](X).$$

This is demonstrable by the following selection of the dependencies in $\tau_{L_N}(P_N)$.

$$\begin{aligned}
\mathit{overloaded}(X) \rightarrow_p \&\mathit{dlC}[L_N, pc_0, mc_0, pr_1, mr_0, \text{“}\mathit{HighTrafficNode}\text{”}](X) \quad &(1)\\
\&\mathit{dlC}[L_N, pc_0, mc_0, pr_1, mr_0, \text{“}\mathit{HighTrafficNode}\text{”}](X) \rightarrow_e pr_1(\text{“}\mathit{wired}\text{”}, X, Y) \quad &(2)\\
pr_1(\text{“}\mathit{wired}\text{”}, X, Y) \rightarrow_p \mathit{connect}(X, Y) \quad &(3)\\
\mathit{connect}(X, Y) \rightarrow_n \mathit{overloaded}(Y) \quad &(4)\\
\mathit{overloaded}(Y) \rightarrow_p \mathit{overloaded}(X) \quad &(5)
\end{aligned}$$

Let $\rightarrow\, = \{(1), (2), (3), (4), (5), \dots\}$ be the dependency relation $\rightarrow_p \cup \rightarrow_e \cup \rightarrow_n$. The transitive closure of $\rightarrow$, $\rightarrow^+$, then contains the dependency

$$\mathit{overloaded}(X) \rightarrow^+ pr_1(\text{“}\mathit{wired}\text{”}, X, Y) \qquad \text{(by (1) and (2))}$$

and the dependency

$$pr_1(\text{“}\mathit{wired}\text{”}, X, Y) \rightarrow^+ \mathit{overloaded}(X) \qquad \text{(by (3), (4), and (5))},$$

hence the strong safeness conditions is violated. Therefore, dlvhex rejects $\tau_{L_N}(P_N)$.

We show now that the HEX-program $\tau_L(P)$ has the same answer sets (possibly extended with auxiliary facts not present in (L, P)) as the dl- or cq-program (L, P). To confirm this, we need the following Lemma.

Lemma 4.7. Let I be a Herbrand interpretation, let $a = \mathrm{DL}[\lambda; q](\vec{c})$ be a ground dl-atom, let I_π be the answer set of $I \cup \pi_L(a)$, let $\lambda = S_1 op_1 p_1, \dots, S_m op_m p_m$ be the input list of a and let $n = \#\lambda$, where $op_i \in \{\uplus, \cup\}$. Then, $I \models_L a$ iff $f_{\&\mathit{dlT}}(I_\pi, L, pc_n, mc_n, pr_n, mr_n, q, \vec{c}) = 1$, where $\&\mathit{dlT}$ is the respective DL external atom for $q(\vec{c})$.

Proof. In the following, let $P = pc_n, pr_n$ and $N = mc_n, mr_n$ be a positive and a negative filter predicate list, respectively, and we denote by $L \models q(c_1, \ldots, c_m)$ that $\langle c_1, \ldots, c_m \rangle \in ans(q(X_1, \ldots, X_m), L)$.

($\Rightarrow$) Suppose $I \models_L a$, i.e., $L \cup \lambda(I) \models q(\vec{c})$. For $op_i = \uplus$, from $p_i(\vec{e}) \in I$ we conclude that for S_i being a concept, $pc_n(\text{"}S_i\text{"}, \vec{e}) \in I_\pi$, otherwise S_i is a role and $pr_n(\text{"}S_i\text{"}, \vec{e}) \in I_\pi$. The case for $op_i = \cupdot$ is similar, but now we conclude that $mc_n(\text{"}S_i\text{"}, \vec{e}) \in I_\pi$ and $mr_n(\text{"}S_i\text{"}, \vec{e}) \in I_\pi$ for S_i being a concept and role, respectively. Thus, $P(I_\pi) \cup N(I_\pi) = \lambda(I)$ and $f_{\&dlT}(I_\pi, L, pc_n, mc_n, pr_n, mr_n, q, \vec{c}) = 1$.

($\Leftarrow$) Now suppose $f_{\&dlT}(I_\pi, L, pc_n, mc_n, pr_n, mr_n, q, \vec{c}) = 1$, hence $L \cup P(I_\pi) \cup N(I_\pi) \models q(\vec{c})$. By minimality of I_π and pc_n, pr_n, mc_n, mr_n are fresh predicate symbols from $\pi_L(a)$, i.e., there is no ground literal $l \in I$ such that one of pc_n, pr_n, mc_n, mr_n is a predicate for l, we derive that for each of such atoms $pc_n(\text{"}S_i\text{"}, \vec{e})$, $pr_n(\text{"}S_i\text{"}, \vec{e})$, $mc_n(\text{"}S_i\text{"}, \vec{e})$, $mr_n(\text{"}S_i\text{"}, \vec{e}) \in I_\pi$, we have a corresponding $p_i(\vec{e}) \in I$. Therefore, $L \cup \lambda(I) \models q(\vec{c})$, which is tantamount to $I \models_L a$. □

In the following, for a cq-program (L, P), let $\pi(P)$ denote the set of HEX-rules $\{\pi_L(r) \mid r \in P\}$, and I_π the answer set obtained from the definite HEX-program $I \cup \pi(P)$. The proofs for the next theorems are located in Appendix A.1.

Theorem 4.8. Let $KB = (L, P)$ be a positive cq-program. Then, I is a minimal model of KB iff I_π is a minimal model of the positive HEX-program $\tau_L(P)$.

Theorem 4.9. Let $KB = (L, P)$ be a cq-program. Then, I is a strong answer set of KB iff I_π is an answer set of $\tau_L(P)$.

Finally, we want to point out that the DL external atoms gives HEX-programs the ability to access more than one DL-KB L. The first input in the external atom input list specifies the desired DL-KB, hence "pure" HEX-programs with DL external atoms can be seen as a combined knowledge base $(\langle L_1, \ldots, L_n \rangle, P)$, where each L_i is a DL-KB occurring in some of the DL external atoms of P, and P is a set of HEX-rules. This setup is mandatory for Ontology Alignment and Matching[7] or Ontology Merging tasks (cf. [Wang et al., 2005] and [Eiter et al., 2006c]).

4.5 Optimization Methods for HEX-Programs

Since calls to the DL-reasoner are an immanent bottleneck in our integration of HEX-programs with DLs, special methods need to be devised in order to optimize the given program for faster and more effective query answering of a DL-KB.

Whatever method may be in use to optimize the interaction between the logic program and the DL-reasoner, the basic optimization aims are to

1. decrease the number of queries to the DL-KB,
2. reduce the amount of transferred data, and
3. possibly decrease the time spent evaluating each query.

To deal with the first goal, we differentiate between program rewriting and query caching techniques, i.e., between syntactic program manipulation and semantic caching, respectively. The former method detects certain patterns in a given program and reduces the number of dl-atoms in it—thus, the total count of queries to a DL-KB decreases before the evaluation

[7]http://ontologymatching.org/

of the program starts—, while the latter does not take a hand in the program, it merely tries to catch queries which are equivalent to some queries previously asked and returns the previous answer to such queries. Here, the number of dl-queries stays the same, but we do not need to ask the DL-reasoner redundantly.

The second optimization target can be attained by applying the preceding techniques: rewriting a program may bring queries which lower the quantity of transferred data—think of a join in a rule which shares a common variable—, and caching query results prevents the transmission of data from the DL-reasoner to the logic programming system over a possibly slow link (the DL-reasoner might not reside on the same host as the logic program solver).

The last goal of our optimization framework effectively aims at reducing the time spent asking the DL-reasoner. Again, all the previous mentioned techniques can be invoked to step things up. But we can add another method, namely knowledge base reusing, which will be covered as last method in this section. In this approach, the dl-plugin tries to detect DL-KBs still in use by the DL-reasoner and operates on them, instead of reloading the ontology. Though less effective than the other methods, it may nevertheless improve the overall performance of a program, since we save the time spent for reclassifying the ontology, especially in the situation where multiple dlvhex instances performs reasoning tasks with RacerPro.

4.5.1 cq-Program optimization

The transformation rules and algorithms for cq-program optimization are subject to Chapter 5 and are covered there in full detail. Here, we just want to mention that due to the presented optimization principles, we gracefully satisfy the need to decrease the number of queries in a program, as well as decrease the time spent evaluating the transformed queries, as is visible from our experimental results.

Moreover, the program rewritings of Chapter 5 naturally apply to their external atoms counterparts, hence we fully support all forms of dl-, cq-, and HEX-programs with DL external atoms. This is backed up by Lemma 4.7. Since DL external atoms basically differ from dl-atoms by allowing to specify the DL-KB as input parameter to the external atom, the optimizations apply only to compatible DL external atoms, i.e., external atoms which share a common DL-KB.

4.5.2 DL caching

As shown in [Eiter et al., 2005a], caching results from the DL-engine is a suitable approach for dealing with redundant queries to the DL-KB, since during the evaluation of cq-programs, dl-atoms may ask queries to the DL-engine. In the worst case, every dl-atom is set up for a call to the DL-reasoner and expects an answer to the issued query. Hence, it is very important to avoid (i) an unnecessary flow of data between the two engines, and (ii) to save time when redundant dl-queries have to be made. Caching methods shown here also applies to the DL external atoms counterparts, hence we stick to the cq-program notion.

We identify two cases for caching answers, the first one deals with pure ground queries, while non-ground queries need other methods for caching results from an external DL-KB. Note that the results of these methods can be equivalently applied to the DL external atoms $\&dlC$, $\&dlR$, and $\&dlDR$, where we explicitly take interpretations I in the oracle functions of the corresponding DL external atoms into account. Thus, we only keep attention to ordinary dl-atoms.

At the present time, the dl-plugin only supports caching of ordinary dl-atoms, as described here. Caching of cq- and ucq-atoms can be imagined, but this implies checking for query containment, which is more costly in general. For more information on query containment, see [Abiteboul et al., 1995, Calvanese et al., 2007b], and cf. [Amiri et al., 2003] for query caching using query containment.

Ground dl-queries For a given cq-program (L, P), external calls must be issued in order to verify whether a given ground dl-atom $\mathrm{DL}[\lambda; Q](\vec{c})$ fulfills $I \models_L \mathrm{DL}[\lambda; Q](\vec{c})$, where I is the current interpretation. In this setting, the caching machinery (see *DL Cache* component in Figure 4.3) exploits properties of monotonic dl-atoms. An easy way to verify that a dl-atom is monotonic is to check the absence of the $\cap\!\!\!-$ operator in its input list. Note that in our implemented system, DL external atoms (and hence dl-atoms) are always monotonic.

The ensuing property from [Eiter et al., 2005a] provides the underpinning of our caching scenery: Given a monotonic ordinary ground dl-atom $a = \mathrm{DL}[\lambda; Q](\vec{c})$ and two interpretations I_1 and I_2 such that $I_1 \subseteq I_2$, monotonicity of a implies that (i) if $I_1 \models_L a$ then $I_2 \models_L a$, and (ii) if $I_2 \not\models_L a$ then $I_1 \not\models_L a$.

With this we are able to build a caching algorithm $\mathit{DLCache}(I, L, a, \mathit{cache})$ in Algorithm 2 for querying and automatic cache maintenance in spirit of the cache maintenance strategy in [Eiter et al., 2005a].

In the following, we will outline this algorithm. The basic idea is to cache the boolean answers to ground dl-atoms a, where only the minimal (resp. maximal) input $\lambda(I)$ for previously supplied interpretations I is stored in the caching subsystem if $I \models_L a$ (resp. $I \not\models_L a$). Exploiting above property, for a monotonic ground dl-atom a we keep a set $\mathit{cache}(a)$ of pairs $\langle \lambda(I), o \rangle$, where $o \in \{\mathit{true}, \mathit{undefined}\}$. If $\langle \lambda(I), \mathit{true} \rangle \in \mathit{cache}(a)$, then we conclude that $J \models_L a$ for each J such that $\lambda(I) \subseteq \lambda(J)$. Dually, if $\langle \lambda(I), \mathit{undefined} \rangle \in \mathit{cache}(a)$, we conclude that $J \not\models_L a$ for each J such that $\lambda(I) \supseteq \lambda(J)$.

Suppose a ground dl-atom $a = \mathrm{DL}[\lambda; Q](\vec{c})$, an interpretation I, and a cache set $\mathit{cache}(a)$ are given as input to our caching system. In order to check whether $I \models_L a$, $\mathit{cache}(a)$ is consulted and updated in method $\mathit{DLCache}(I, L, a, \mathit{cache})$ of Algorithm 2.

Non-ground dl-queries Caching strategies for non-ground queries over DL-KBs is possible too, but we cannot exploit the dl-atom monotonicity property as in the ground situation. Therefore, we restrict our caching strategy to the case where $\lambda(I_1) = \lambda(I_2)$ and save the whole answers to a dl-query.

For each ordinary non-ground dl-atom $a(\vec{t}) = \mathrm{DL}[\lambda; Q](\vec{t})$, a set $\mathit{cache}(a(\vec{t}))$ of pairs $\langle \lambda(I), a{\downarrow}(\lambda(I)) \rangle$ is maintained, where $a{\downarrow}(I)$ is the set of answers to $Q(\vec{t})$, i.e., the set of ground tuples $\vec{c}$ such that $I \models_L a(\vec{c})$. Whenever for some interpretation I, $a{\downarrow}(I)$ is needed, then $\mathit{cache}(a)$ is looked up for some pair $\langle J, a{\downarrow}(J) \rangle$ such that $\lambda(I) = J$.

More general approaches to the one given here can be conceived, such as for two dl-atoms $\mathrm{DL}[\lambda_1; Q](\vec{t})$ and $\mathrm{DL}[\lambda_2; Q](\vec{t})$, we can maintain the cache similarly whenever $\lambda_1 \doteq \lambda_2$; this applies to the ground case of our caching system too.

4.5.3 Knowledge base reusing

Due to the clear semantic separation of the logic programming and the DL part in the dl-plugin, we may reuse DL-KBs loaded in a previous evaluation session. Obviously, this is only possible if the DL-reasoner supports running in “server mode”, i.e., the DL-reasoner process runs independent of dlvhex and waits for queries. This has the effect that we do not

Algorithm 2: *DLCache*$(I, L, a, cache)$: cache querying and maintenance

```
Input: Interpretation $I$, DL-KB $L$, ground dl-atom $a$, and DL cache $cache$
Result: $I \models_L a$?
choose $\langle J, o \rangle$ from $cache(a)$
if $(o = true \land J \subseteq \lambda(I)) \lor (o = undefined \land J \supseteq \lambda(I))$ then return $o$
else
    if $I \models_L a$ then
        $o = true$
        foreach $\langle J, o \rangle \in cache(a)$ do
            if $\lambda(I) \subset J$ then remove $\langle J, o \rangle$ from $cache(a)$
        end
    else
        $o = undefined$
        foreach $\langle J, o \rangle \in cache(a)$ do
            if $\lambda(I) \supset J$ then remove $\langle J, o \rangle$ from $cache(a)$
        end
    end
    add $\langle \lambda(I), o \rangle$ to $cache(a)$
    return $o$
end
```

have to wait for the DL-reasoner to classify the TBox of the desired ontology again every time we start the dlvhex reasoning process (see also the branch after the "Open DL-KB" state in the query processing Figure 4.5).

This optimization method only works under the assumption, that different Ontologies have different associated names. Suppose a previously loaded Ontology has been updated in the data source and the associated name of this Ontology did not change. Since there is no easy way to decide whether we need to reload the DL-KB in the DL-reasoner, we still use the outdated preloaded Ontology of the DL-reasoner, which may be incompatible with the program, or gives unexpected results. This is circumvented by forcing the DL-reasoner to reload the DL-KB in every dlvhex reasoning session.

4.6 Plugin Usage

The dl-plugin interfaces HEX-programs with OWL ontologies by using a DL reasoning system. Currently, the DL-reasoner RacerPro is supported. It provides five external atoms, roughly divided in (i) querying atoms, i.e., external atoms which requests answers from the DL-reasoner to a given query, and (ii) the DL-KB consistency checking external atom, a boolean atom which decides whether augmenting a DL-KB with the given input parameters results in a consistent DL-KB. The atoms of the first category support query answering to concept (C) and role (R and DR) queries, conjunctive queries (CQ), and union of conjunctive queries (UCQ).

4.6.1 Command line options

dlvhex advertises its supplied command line options to the plugins, such that each plugin can pick its particular option for further processing and state modification.

The dl-plugin accepts the next command line arguments:

`L`	URI or file path of the OWL ontology L
`a`	name of a binary predicate whose extension denotes addition to a concept
`b`	name of a binary predicate whose extension denotes addition to the complement of a concept
`c`	name of a ternary predicate whose extension denotes addition to a role
`d`	name of a ternary predicate whose extension denotes addition to the complement of a role

Table 4.1: Common input parameters of all dl-plugin external atoms

• `--ontology=URI`: In cq-programs (L, P), set the corresponding DL-KB L to `URI`, where `URI` is the location of an OWL Ontology in URI syntax. See 4.6.3 for more information about cq-programs.

• `--kb-reload`: With this option, we force a reload of a previously loaded DL-KB.

• `--dlsetup=ARG[,ARG]*`: Setup some DL-reasoner options according to the supplied list of arguments `ARG`, which may be `-una` for disabling the Unique Name Assumption in the DL-reasoner.

• `--dlopt=MOD[,MOD]*`: Setup particular optimization features according to the supplied list of modifiers `MOD`, which may be `-push` for disabling push optimizations and `-cache` for disabling the DL-Cache.

• `--dldebug=LEVEL`: For debugging purposes, set `LEVEL` accordingly to increase the verbosity of the log messages during query evaluation.

4.6.2 External atoms

The dl-plugin supports all DL external atoms and the *&dlConsistent* external atom, which will be defined later.

All external atoms share common input parameters. Their intended meaning is specified in Table 4.1. A more detailed explanation of all external atoms with examples, the concrete syntax for queries q in DL external atoms, and the output list $X_1, \ldots, X_n$ is now subject of the remainder of this section.

Concept queries Queries to concepts are stated using the external atom

$$\texttt{\&dlC[L,a,b,c,d,q](X)}, \tag{4.1}$$

where `q` is a concept name and `X` is a term. If the external atom has a non-ground output, i.e., `X` is a variable, then (4.1) retrieves all known members of concept `q`. Otherwise, if `X` is an individual, then (4.1) holds iff `X` is an instance of concept `q`.

Example 4.10. The following rule expresses a simple concept query:

```
wine(X) :- &dlC["wine.rdf",a,b,c,d,"Wine"](X).
```

Provided that *a*, *b*, *c*, and *d* do not occur elsewhere in the HEX-program, this rule would do nothing else than putting all members of concept *Wine* of the wine ontology `"wine.rdf"`[8] into the predicate *wine*.

[8]see `http://www.w3.org/TR/owl-guide/wine.rdf`

The term "*Wine*" is expressed in RDF and has thus an XML namespace attached to it. Since the concept *Wine* uses the default namespace name

`http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#`, (4.2)

we simply refer to the concept of all wines as is. This would not be possible, if `Wine` is in the scope of a different namespace than the default namespace (4.2). Consider an ontology, where the concept of all wines is defined as follows:

```
<owl:Class rdf:ID="vin:Wine"/>
```

Here, the term `"Wine"` is bound to the XML namespace `vin`. To refer to the concept `vin:Wine`, which is short for the RDF/XML URI

`http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Wine`, (4.3)

we may use two different methods for doing so. First, we can use the fully expanded concept name (4.3) in the query part of the external atoms, or second, we introduce a namespace in the HEX-program itself. This is accomplished by adding a namespace declaration to the program. Take, for instance, the program in Example 4.10. Adding the XML namespace `vin` to it results in the program

```
#namespace(vin,"http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#")
wine(X) :- &dlC["wine.rdf",a,b,c,d,"vin:Wine"](X).
```

Note that we may define an arbitrary namespace prefix for the namespace URI (4.2). The next two examples show the basic usage pattern of concept querying external atoms in HEX-programs.

Example 4.11. Now imagine we want to extend the wine concept by a new individual currently not known to be a wine. We use the concept augmentation mechanisms of the *&dlC* atom.

```
wine(X) :- &dlC["wine.rdf",w,b,c,d,"Wine"](X).
w("Wine","Uhudler").
```

In the preceding example, we add the individual *Uhudler* to the concept *Wine*. The *&dlC* atom expects a predicate name as second parameter; in this case *w*. This predicate must be binary and its first argument denotes the concept to be extended and the second the actual individual to be added to the concept.

This sort of augmentation of DL-KBs can be extended to involve even more flexible queries. In the next program, we supplement *Uhudler* to *Wine* and all italian redwines to the concept *RedWine*.

```
wine(X) :- &dlC["wine.rdf",w,b,c,d,"Wine"](X).
w("Wine","Uhudler").
w("RedWine",X) :- redwine(X), grows(X,italy).
```

Example 4.12. Similarly, we may increase the ABox of an ontology by role axioms. The third input parameter of a *&dlC* atom specifies the ternary predicate name.

```
wine(X) :- &dlC["wine.rdf",a,b,l,d,"Wine"](X).
l(locatedIn,X,Y) :- grows(X,Y).
```

In this program, before we query the concept *Wine*, we extend concept *locatedIn* by predicate *grows* with the aid of `l`.

Role queries Role querying is accomplished using the external atom

$$\texttt{\&dlR[L,a,b,c,d,q](X,Y)}, \tag{4.4}$$

where q is a role name and X and Y are terms. If the external atom has a non-ground output, i.e., both X and Y are variables, then (4.4) retrieves all known pairs of role q. If both X and Y is are individuals, then (4.4) holds iff (X,Y) is an instance of role q. If only one of X and Y is ground, then we retrieve all fillers for respective individual in the role.

OWL Datatype Properties queries are subject to a different atom.

$$\texttt{\&dlDR[L,a,b,c,d,q](X,Y)} \tag{4.5}$$

is basically the same as the *&dlR* atom, but q is a Datatype Property here.

The input mechanism of role querying atoms are similar to the input mechanism of *&dlC* external atoms.

Example 4.13. The next example retrieves all pairs $(X, \mathit{NewZealandRegion})$ from role *locatedIn* right after the ABox has been increased by the extension of l, i.e., all *grows* are added to *locatedIn*.

```
nzwine(X) :- &dlR["wine.rdf",a,l,c,d,locatedIn](X,"NewZealandRegion").
l(locatedIn,X,Y) :- grows(X,Y).
```

Conjunctive and union of conjunctive queries The next two atoms provide support for conjunctive and union of conjunctive queries:

$$\texttt{\&dlCQ}[\mathtt{L},\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{cq}](\mathtt{X}_1,\dots,\mathtt{X}_\mathtt{n}), \tag{4.6}$$

$$\texttt{\&dlUCQ}[\mathtt{L},\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{ucq}](\mathtt{X}_1,\dots,\mathtt{X}_\mathtt{n}), \tag{4.7}$$

where cq and ucq is a conjunctive query and a union of conjunctive queries, respectively. Both output lists are formed of an n-ary tuple $(\mathtt{X}_1,\dots,\mathtt{X}_\mathtt{n})$, where each $\mathtt{X}_\mathtt{i}$ is a variable. Note that in general, HEX-programs allow ground terms as arguments in external atoms, but ground terms in the output of *&dlCQ* and *&dlUCQ* atoms are useless. Therefore, we omit the case where some $\mathtt{X}_\mathtt{i}$ may be ground.

A conjunctive query cq is a query of form

$$\mathtt{Q}_1(\tilde{\mathtt{X}_1}),\mathtt{Q}_2(\tilde{\mathtt{X}_2}),\dots,\mathtt{Q}_\mathtt{n}(\tilde{\mathtt{X}_\mathtt{n}}), \tag{4.8}$$

where for $1 \leq \mathtt{i} \leq \mathtt{n}$ each $\mathtt{Q}_\mathtt{i}$ is a concept or role name and $\tilde{\mathtt{X}_\mathtt{i}}$ a single term or a pair of terms.

Similarly, a union of conjunctive queries ucq is a query of form

$$\mathtt{cq}_1 \ \mathtt{v} \ \mathtt{cq}_2 \ \mathtt{v} \ \cdots \ \mathtt{v} \ \mathtt{cq}_\mathtt{n}, \tag{4.9}$$

where for $1 \leq \mathtt{i} \leq \mathtt{n}$ each $\mathtt{cq}_\mathtt{i}$ is a conjunctive query.

The examples below show how to use query DL-KBs using the *&dlCQ* and *&dlUCQ* external atoms.

Example 4.14. Consider the following rule issuing a conjunctive query over the wine ontology. As result, we retrieve all dry wines from the Burgundy region in predicate *bdw*.

```
#namespace(vin,"http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#")
bdw(X) :- &dlCQ["wine.rdf",a,b,c,d,"Burgundy(X),hasSugar(X,vin:Dry)"](X).
```

Example 4.15. The next rule retrieves all white wines or pasta dishes; it is a convenient example for expressing UCQs.

```
#namespace(food,"http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#")
wp(X) :- &dlUCQ["wine.rdf",a,b,c,d,"WhiteWine(X) v food:Pasta(X)"](X).
```

DL-KB consistency The last external atom is *&dlConsistent*, which tests the given DL-KB for consistency under the specified extensions. It is of form

$$\texttt{\&dlConsistent[L,a,b,c,d].} \tag{4.10}$$

If `L` is consistent after possibly augmenting the ABox according to the input list, the atom (4.10) evaluates to true, otherwise false.

Example 4.16. The program

```
u("Wine","Uhudler").
:- not &dlConsistent["wine.rdf",u,u,c,d].
```

has no answer set, since we augment the ABox of *wine.rdf* by the axioms $Wine(Uhudler)$ and $\neg Wine(Uhudler)$.

4.6.3 cq-programs

cq-programs, as defined in Section 3, can be evaluated using dlvhex and the dl-plugin. Section 4.4 gives a partial equivalence between cq-atoms and the external atoms defined in the dl-plugin.

In order to process cq-programs with dlvhex, the concrete syntax for dl-atoms of form (3.3) is

```
DL[S1 op1 p1,...,Sm opm pm; q](X1,...,Xn)
```

where `q(X1,...,Xn)` is a dl- or (u)cq-query and `opi` is `+=` and `-=` for $op_i = \uplus$ and $\cup\!\!\!\!-$, resp. Since we do not encode the ontology in the program, we must add the option `--ontology=URL` to the dlvhex command line arguments, where `URL` is a file or a URL to an OWL Ontology.

Example 4.17. An simple example of a cq-program is (L, P), where L is the wine ontology in `http://www.w3.org/TR/owl-guide/wine.rdf` and P is the rule

```
wine(X) :- DL[Wine](X).
```

(L, P) is equivalent to the program in Example 4.10, provided that above rule is in a file called `wine.dlp` and we call dlvhex as follows:

```
$ dlvhex --ontology=http://www.w3.org/TR/owl-guide/wine.rdf wine.dlp
```

Similarly, we encode Example 4.11 as cq-program

```
wine(X) :- DL[Wine += w; Wine](X).
w("Uhudler").
```

and get the same extensions in `wine` as in the HEX-rule, given that we call the dlvhex program in the same way as above.

On my planet, to rest is to rest—to cease using energy. To me, it is quite illogical to run up and down on green grass, using energy, instead of saving it.

—Spock in Star Trek, *Shore Leave*, stardate 3025.2

5 Optimization of cq-Programs

In this chapter we present optimization techniques and algorithms as well as experimental results for optimizing cq-programs introduced in [Eiter et al., 2007a] and presented in Chapter 3. These results provide the theoretical framework for the optimization implementation in Chapter 4.

Motivated by the potential optimization aspect of conjunctive and union of conjunctive queries over DL-KBs in cq-programs, we now focus on the following contributions.

• In Section 5.1 we present a suite of equivalence-preserving transformation rules, by which rule bodies and rules involving (u)cq-atoms can be rewritten. Based on these rules, we then describe algorithms which transform a given cq-program (L, P) into an equivalent, optimized cq-program (L, P') in Section 5.2.

In a sense, they constitute a first example where the investigation on the common fragment between a rule language and a description logics formalism has a practical revenue in terms of evaluation times.

• We report an experimental evaluation of the rewriting techniques in Section 5.3, based on the prototype implementation of cq-programs presented in Chapter 4 using dlvhex [Eiter et al., 2005b] and RacerPro [Haarslev and Möller, 2001]. It shows the effectiveness of the techniques, and that significant performance increases can be gained. The experimental results are interesting in their own right, since they shed light on combining conjunctive query results from a DL-reasoner.

5.1 Rewriting Rules for cq- and ucq-Atoms

As shown in Example 3.3, in cq-programs we might have different possibilities for defining the same query. Indeed, the rules r and r' there are equivalent over any description logic knowledge base L. However, the evaluation of r' might be implemented by performing the join between *parent* and *hates* on the DL side in a single call to a DL-reasoner, while r can be evaluated performing the join on the logic program side, over the results of two calls to the DL-reasoner. In general, making more calls is more costly, and thus r' may be preferable from a computational point of view. Moreover, the size of the result transferred by the single call in rule r' is smaller than the results of the two calls.

Towards exploiting such rewriting, we present some transformation rules for replacing a rule or a set of rules in a cq-program with another rule or set of rules, while preserving the semantics of the program. By means of (repeated) rule application, we transform the program into another, equivalent program, which we consider in the next section. Such repeated application process is dealt with in the algorithms presented in Section 5.2. Put together, these algorithms form a software component for rewriting cq-programs. Indeed, a rewriting module is conceivable, which rewrites a given cq-program (L, P) into a refined, equivalent cq-program (L, P'), which can be evaluated more efficiently. The rules are compactly summarized in Table 5.1, and will be discussed, with the necessary details, in the sequel. Note that as for rule application, any ordinary dl-atom $\mathrm{DL}[\lambda; Q](\vec{t})$, where $\vec{t}$ is a non-empty list of terms, is equivalent to the cq-atom $\mathrm{DL}[\lambda; Q(\vec{t})](\vec{X})$, where $\vec{X} = \mathrm{vars}(\vec{t})$.

In the rewriting rules, the input lists λ_1 and λ_2 are assumed to be semantically equivalent (denoted $\lambda_1 \doteq \lambda_2$), that is, $\lambda_1(I) = \lambda_2(I)$, for every Herbrand interpretation I. This means that λ_1 and λ_2 modify the same concepts and roles with the same predicates in the same way; this can be easily recognized (in fact, in linear time). More liberal but more expensive notions of equivalence, taking L and/or P into account, might be considered.

5.1.1 Query Pushing (A)

By this rule, cq-atoms $\mathrm{DL}[\lambda_1; cq_1](\vec{Y_1})$ and $\mathrm{DL}[\lambda_2; cq_2](\vec{Y_2})$ in the body of a rule (A1) can be merged.

$$r: \ H \leftarrow \mathrm{DL}[\lambda_1; cq_1](\vec{Y_1}), \mathrm{DL}[\lambda_2; cq_2](\vec{Y_2}), B. \tag{A1}$$

$$r': \ H \leftarrow \mathrm{DL}\big[\lambda_1; cq_1' \cup cq_2'\big](\vec{Y_1} \cup \vec{Y_2}), B. \tag{A2}$$

In rule (A2), cq_1' and cq_2' are constructed by renaming variables in cq_1 and cq_2 as follows. Let $\vec{Z_1}$ and $\vec{Z_2}$ be the non-distinguished variables of cq_1 and cq_2, respectively. Rename each $X \in \vec{Z_1}$ occurring in cq_2 and each $X \in \vec{Z_2}$ occurring in cq_1 to a fresh variable. Then $cq_1' \cup cq_2'$ is the CQ given by all the atoms in both CQs.

Example 5.1. The rule

$$a \leftarrow \mathrm{DL}[R_1(X, Y), R_2(Y, Z)](X), \mathrm{DL}[R_3(X, Y)](X, Y)$$

is equivalent to the rule

$$a \leftarrow \mathrm{DL}\big[R_1(X, Y'), R_2(Y', Z), R_3(X, Y)\big](X, Y)$$

over all DL-KBs.

Query Pushing can be similarly done when cq_1 and cq_2 are UCQs; here, we simply distribute the subqueries and form a single UCQ.

5.1.2 Variable Elimination (B)

Suppose an output variable X of a cq-atom in a rule r of form (B1a) or (B1b) occurs also in an atom $X = t$. Assume that t is different from X and that, in case of form (B1a) the underlying DL-KB is under Unique Name Assumption (UNA) whenever t is an output variable. Then, we eliminate X from r as follows. Standardize the non-output variables of cq-atoms apart from the other variables in r, and replace uniformly X with t in cq, B, and H; let $cq_{X/t}$, $B_{X/t}$, and $H_{X/t}$ denote the respective results. Remove X from the output $\vec{Y}$

Query Pushing

$$r: \; H \leftarrow \mathrm{DL}[\lambda_1; cq_1](\vec{Y_1}), \mathrm{DL}[\lambda_2; cq_2](\vec{Y_2}), B. \qquad \text{(A1)}$$

$$r': \; H \leftarrow \mathrm{DL}\big[\lambda_1; cq_1' \cup cq_2'\big](\vec{Y_1} \cup \vec{Y_2}), B. \qquad \text{(A2)}$$

where $\lambda_1 \doteq \lambda_2$.

Variable Elimination

$$r_1: \; H \leftarrow \mathrm{DL}[\lambda_1; cq \cup \{X = t\}](\vec{Y}), B. \qquad \text{(B1a)}$$

$$r_2: \; H \leftarrow \mathrm{DL}[\lambda_1; cq](\vec{Y}), X = t, B. \qquad \text{(B1b)}$$

$$r': \; H_{X/t} \leftarrow \mathrm{DL}\big[\lambda_2; cq_{X/t}\big](\vec{Y} \setminus \{X\} \cup \omega(t)), B_{X/t}. \qquad \text{(B2)}$$

where $\lambda_1 \doteq \lambda_2$, $X \in \vec{Y}$, $\cdot_{X/t}$ denotes replacement of variable X by t, and $\omega(t) = \{Z\}$ if t is a variable Z and $\omega(t) = \emptyset$ otherwise.

Inequality Pushing

$$r: \; H \leftarrow \mathrm{DL}[\lambda_1; cq](\vec{Y}), X \neq t, B. \qquad \text{(C1)}$$

$$r': \; H \leftarrow \mathrm{DL}[\lambda_2; cq \cup \{X \neq t\}](\vec{Y}), B. \qquad \text{(C2)}$$

where $\lambda_1 \doteq \lambda_2$ and $X \in \vec{Y}$. If t is a variable, then also $t \in \vec{Y}$.

Fact Pushing

$$\bar{P} = \left\{ \begin{array}{l} f(\vec{c_1}), f(\vec{c_2}), \ldots, f(\vec{c_l}), \\ \;H \leftarrow \;\; \mathrm{DL}[\lambda_1; \bigvee_{i=1}^{r} cq_i](\vec{Y}), f(\vec{Y'}), B. \end{array} \right\} \qquad \text{(D1)}$$

$$\bar{P}' = \left\{ \begin{array}{l} f(\vec{c_1}), f(\vec{c_2}), \ldots, f(\vec{c_l}), \\ \;H \leftarrow \;\; \mathrm{DL}\Big[\lambda_2; \bigvee_{i=1}^{r} \Big(\bigvee_{j=1}^{l} cq_i \cup \{\vec{Y'} = \vec{c_j}\}\Big)\Big](\vec{Y}), B. \end{array} \right\} \qquad \text{(D2)}$$

where $\lambda_1 \doteq \lambda_2$, $\vec{c_j}$ are ground, $\vec{Y'} \subseteq \vec{Y}$.

Let H, H', H_i be heads, B, B', B_i be bodies, and r be a rule of form $H \leftarrow a(\vec{Y}), B$.

Unfolding

$$\bar{P} = \{r\} \cup \{H' \vee a(\vec{Y'}) \leftarrow B'.\} \qquad \text{(E1)}$$

$$\bar{P}' = \bar{P} \cup \{H'\theta \vee H\theta \leftarrow B'\theta, B\theta.\} \qquad \text{(E2)}$$

where θ is the mgu of $a(\vec{Y})$ and $a(\vec{Y'})$ (thus $a(\vec{Y}\theta) = a(\vec{Y'}\theta)$).

Complete Unfolding

$$P = Q \cup \{r\} \cup \{\; r_i: \;\; H_i \vee a(\vec{Y_i}) \leftarrow B_i. \qquad (1 \leq i \leq l) \;\} \qquad \text{(F1)}$$

$$P' = (P \setminus \{r\}) \cup \{\; r_i': \;\; H_i\theta_i \vee H\theta_i \leftarrow B_i\theta_i, B\theta_i. \qquad (1 \leq i \leq l) \;\} \qquad \text{(F2)}$$

where Q has no rules of form r, r_i, no $a(\vec{Z}) \in H_i$ is unifiable with $a(\vec{Y})$, and θ_i is the mgu of $a(\vec{Y})$ and $a(\vec{Y_i})$ (thus $a(\vec{Y}\theta_i) = a(\vec{Y_i}\theta_i)$).

Table 5.1: Equivalences ($H = a_1 \vee \cdots \vee a_k$, $B = b_1, \ldots, b_m, \mathrm{not}\, b_{m+1}, \ldots, \mathrm{not}\, b_n$)

and, if t is a variable Z, add Z to them; the resulting rule r', in (B2) is then equivalent to the rule r_1 in (B1a) or to the rule r_2 in (B1b).

$$r_1: \; H \leftarrow \mathrm{DL}[\lambda_1; cq \cup \{X = t\}](\vec{Y}), B. \qquad \text{(B1a)}$$

$$r_2: \; H \leftarrow \mathrm{DL}[\lambda_1; cq](\vec{Y}), X = t, B. \qquad \text{(B1b)}$$

$$r': \; H_{X/t} \leftarrow \mathrm{DL}\big[\lambda_2; cq_{X/t}\big](\vec{Y} \setminus \{X\} \cup \omega(t)), B_{X/t}. \qquad \text{(B2)}$$

By repeated application of this rule, we may eliminate multiple output variables of a cq-atom. Note that variables X in equalities $X = t$ not occurring in any output list can always be eliminated by simple replacement.

Example 5.2. The rule

$$r : a_1(X) \vee a_2(Y) \leftarrow \mathrm{DL}[R(X,Z), C(Y), X = Y](X,Y), b(Y)$$

and rule

$$r' : a_1(Y) \vee a_2(Y) \leftarrow \mathrm{DL}[R(Y,Z), C(Y)](Y), b(Y)$$

have the same outcome on every DL-KB L. Here, r' should be preferred due to the lower arity of its cq-atom.

Similarly, the rule $a(X,Y) \leftarrow \mathrm{DL}[R(X,Z), C(Y), Y = c](X,Y), b(Y)$ is simplified to the rule $a(X,c) \leftarrow \mathrm{DL}[R(X,Z), C(c)](X), b(c)$.

5.1.3 Inequality Pushing (C)

If the DL-engine is used under the UNA and supports inequalities in the query language, we easily rewrite rules with inequality ($\neq$) in the body by pushing it to the cq-query. A rule of form (C1) can be replaced by (C2).

$$r :\ H \leftarrow \mathrm{DL}[\lambda_1; cq](\vec{Y}), X \neq t, B. \tag{C1}$$

$$r' :\ H \leftarrow \mathrm{DL}[\lambda_2; cq \cup \{X \neq t\}](\vec{Y}), B. \tag{C2}$$

Example 5.3. Consider the rule

$$\begin{aligned} r :\ \mathit{bigwinery}(M) \leftarrow\ & \mathrm{DL}[\mathit{Wine}](W_1), \mathrm{DL}[\mathit{Wine}](W_2), W_1 \neq W_2, \\ & \mathrm{DL}[\mathit{hasMaker}](W_1, M), \mathrm{DL}[\mathit{hasMaker}](W_2, M). \end{aligned}$$

Here, we want to know all wineries producing at least two different wines. We rewrite r, by Query and Inequality Pushing, to the rule

$$r' :\ \mathit{bigwinery}(M) \leftarrow \mathrm{DL}\left[\begin{array}{c} \mathit{Wine}(W_1), \mathit{Wine}(W_2), W_1 \neq W_2 \\ \mathit{hasMaker}(W_1, M), \mathit{hasMaker}(W_2, M) \end{array}\right](M, W_1, W_2).$$

A similar rule works for a ucq-atom $\mathrm{DL}[\lambda; \mathit{ucq}](\vec{Y})$ in place of $\mathrm{DL}[\lambda; \mathit{cq}](\vec{Y})$. In that case, we have to add $\{X \neq t\}$ to each cq_i in $\mathit{ucq} = \bigvee_{i=1}^{m} \mathit{cq}_i$.

Observe that, in general, answering CQs becomes undecidable when $\neq$ appears in a (U)CQ (see [Calvanese et al., 2007b]); [Rosati, 2007b] identifies various DLs for which (U)CQs with inequalities remain decidable.

5.1.4 Fact Pushing (D)

Suppose we have a program with "selection predicates," i.e., facts which serve to select a specific property in a rule. We can push such facts into a ucq-atom and remove the selection atom from the rule body.

Example 5.4. Consider the program P, where we only want to know the children of *joe* and *jill*.

$$\begin{aligned} & f(\mathit{joe}).\quad f(\mathit{jill}). \\ & \mathit{fchild}(Y) \leftarrow \mathrm{DL}[\mathit{isFatherOf}](X,Y), f(X). \end{aligned}$$

We may rewrite the program to a more compact one with the help of ucq-atoms.

$$\begin{aligned} & f(\mathit{joe}).\quad f(\mathit{jill}). \\ & \mathit{fchild}(Y) \leftarrow \mathrm{DL}\left[\begin{array}{c} \{\mathit{isFatherOf}(X,Y), X = \mathit{joe}\} \vee \\ \{\mathit{isFatherOf}(X,Y), X = \mathit{jill}\} \end{array}\right](X,Y). \end{aligned}$$

Such a rewriting makes sense in situations were *isFatherOf* has many tuples and thus would lead to transfer all known father child relationships.

The program $\bar{P}$ in (D1) can be rewritten to $\bar{P}'$ in (D2).

$$\bar{P} = \left\{ \begin{array}{l} f(\vec{c_1}), f(\vec{c_2}), \ldots, f(\vec{c_l}), \\ H \leftarrow \mathrm{DL}[\lambda_1; \bigvee_{i=1}^{r} cq_i](\vec{Y}), f(\vec{Y'}), B. \end{array} \right\} \tag{D1}$$

$$\bar{P}' = \left\{ \begin{array}{l} f(\vec{c_1}), f(\vec{c_2}), \ldots, f(\vec{c_l}), \\ H \leftarrow \mathrm{DL}\Big[\lambda_2; \bigvee_{i=1}^{r} \Big(\bigvee_{j=1}^{l} cq_i \cup \{\vec{Y'} = \vec{c_j}\}\Big)\Big](\vec{Y}), B. \end{array} \right\} \tag{D2}$$

In general, a cq-program P such that $\bar{P} \subseteq P$ and f does not occur in heads of rules in $P \setminus \bar{P}$ can be rewritten to $(P \setminus \bar{P}) \cup \bar{P}'$.

5.1.5 Unfolding (E) and Complete Unfolding (F)

Unfolding rules is a standard method for partial evaluation of ordinary disjunctive logic programs under answer-set semantics, cf. [Sakama and Seki, 1997] and Section 2.7. It can be also applied in the context of cq-programs, with no special adaptation. After folding rules with dl-atoms in their body into other rules, subsequent Query Pushing might be applied. In this way, inference propagation can be shortcut.

Accordingly, for a rule r of form

$$H \leftarrow a(\vec{Y}), B,$$

where $H = a_1 \vee \cdots \vee a_k$ and $B = b_1, \ldots, b_m, \mathrm{not}\, b_{m+1}, \ldots, \mathrm{not}\, b_n$, rules of form (E1) can be equivalently rewritten to the rules (E2) as follows. Suppose r and r' do not share variables (otherwise, rename variables in r' first). Let θ be a *most general unifier* of $a(\vec{Y})$ and $a(\vec{Y'})$; then, apply θ to r without $a(\vec{Y})$, and then we add $H'\theta$ in the head and $B'\theta$ in the body.

$$\bar{P} = \{r\} \cup \{r' : H' \vee a(\vec{Y'}) \leftarrow B'.\} \tag{E1}$$

$$\bar{P}' = \bar{P} \cup \{H'\theta \vee H\theta \leftarrow B'\theta, B\theta.\} \tag{E2}$$

Similarly, if r is in P of (F1) such that Q does not contain rules of form r or r_i, $1 \leq i \leq l$, then we replace P by the equivalent P' in (F2) using the unifiers θ_i for each r_i.

$$P = Q \cup \{r\} \cup \{\ r_i :\ \ H_i \vee a(\vec{Y_i}) \leftarrow\ \ B_i. \qquad (1 \leq i \leq l)\ \} \tag{F1}$$

$$P' = (P \setminus \{r\}) \cup \{\ r_i' :\ \ H_i\theta_i \vee H\theta_i \leftarrow\ \ B_i\theta_i, B\theta_i. \qquad (1 \leq i \leq l)\ \} \tag{F2}$$

Notice that if $\bar{P} \subseteq P$, then we are free to add the rule r' to P without changing its answer sets. Moreover, in Complete Unfolding, we remove r from P after having unfolded $a(\vec{Y})$ in the body of r in all possible ways to P'.

5.1.6 Equivalence theorems

The following results state that the above rewritings preserve equivalence. Let $P \equiv_L Q$ denote that (L, P) and (L, Q) have the same answer sets. See Appendix A.2 for the proofs of the theorems.

Theorem 5.5. Let r and r' be rules of form (Θ1) and (Θ2), respectively, $\Theta \in \{A, B, C\}$. Let (L, P) be a cq-program with $r \in P$. Then, $P \equiv_L (P \setminus \{r\}) \cup \{r'\}$.

Theorem 5.6. Let $\bar{P}$ and $\bar{P}'$ be rule sets of form (Θ1) and (Θ2), respectively, $\Theta \in \{D, E\}$. Let (L, P) be a cq-program such that $\bar{P} \subseteq P$. Then, $\bar{P} \equiv_L \bar{P}'$ and $P \equiv_L (P \setminus \bar{P}) \cup \bar{P}'$.

Theorem 5.7. Let P and P' be rule sets of form (F1) and (F2). Then, $P \equiv_L P'$.

5.2 Rewriting Algorithms

Based on the results above, we describe algorithms which combine them into a single module for optimizing cq-programs. The optimization process takes several steps. In each step, a special rewriting algorithm works on the result handed over by the preceding step. Note that, in general, some of the rewriting rules might eliminate some predicate name from a given program. This might not be desired if such predicate names play the role of output predicates. Indeed, it is usual that a program P contains auxiliary rules conceived for importing knowledge from an ontology, or to compute intermediate results, while important information, from the user point of view, is carried by output predicates. We introduce thus a set F of *filter* predicates which are explicitly preserved from possible elimination.

5.2.1 Rewriting with Unfolding and Fact Pushing

The first step performs unfolding, taking into account the content of F. That is, only literals with a predicate from F are kept.

Algorithm 3 uses the function $\mathit{factpush}(P)$ for Fact Pushing. This function tries to turn a program P into a more efficient one by merging rules according to the equivalences in Section 5.1.4. The algorithm also combines filtering and unfolding using $\mathit{unfold}(a, r, r')$, which takes two rules r and r' and returns the unfolding of r' with r w.r.t. a literal a. Note that $\mathit{do\_unfold}(a, r, r', P)$ is a generic function for deciding whether the unfolding of a rule r in r' w.r.t. a given program P and a literal a can be done (or is worth being done); this may be carried out, e.g., using a cost model (as we will see later in Section 5.2.3). *do_unfold* may also use, e.g., an internal counter for the numbers of iterations or rule unfoldings, and return false if a threshold is exceeded. Also, complete unfolding cannot take place if more than one atom in the head of r' can unify with a. The function $\mathit{filter}(P, F)$ eliminates rules which have no influence on the filtered output. Such rules are those of form $H \leftarrow B$ where H is nonempty and has no predicate from F and no literal a unifiable either (i) with some literal in the body of a rule from P, or (ii) with some literal in a disjunctive rule head in P, or (iii) with the opposite of some literal in a rule head in P.

Theorem 5.8. For a cq-program (L, P) and filter F, $P \equiv_L \mathit{merge}(P, F)$ w.r.t. F.

Proof. See Appendix A.2. □

5.2.2 Rewriting with Pushing and Variable Elimination

After the unfolding process, we may use Algorithm 4 for optimizing all the different kinds of queries in P. Here, $\mathit{push}(a_1, a_2)$ takes any combination of two dl-, cq-, and ucq-atoms and generates an optimized cq- or ucq-atom. Similar to *do_unfold* in Algorithm 3, $\mathit{do\_push}(a_1, a_2)$ is a generic function to decide whether the Query/Inequality Pushing should take place, i.e., it checks compatibility of the input lists of the atoms and whether pushing of a_1 and a_2 yields a more efficient query.

The last part in this algorithm eliminates variables in the output of dl-atoms according to the Variable Elimination rule.

Theorem 5.9. For every cq-program (L, P), $P \equiv_L \mathit{RuleOptimizer}(P)$.

Proof. See Appendix A.2. □

Algorithm 3: *merge*(P, F): Merge cq-rules in program P w.r.t. F

```
Input: Program $P$, Filter $F = \{p_1, \dots, p_n\}$
Result: Unfolded program $P$
repeat
    $P^l = P = factpush(P)$
    $C = \{a, a' \mid \exists\, r, r' \in P : a' \in H(r'), a \in B^+(r),$ and $a'$ unifiable with $a\}$
    if $C \neq \emptyset$ then
        choose $a \in C$
        $P' = \emptyset$
        $R_H = \{r \in P \mid a$ unifies with $a' \in H(r)\}$
        $R_B = \{r \in P \mid a$ unifies with $a' \in B^+(r)\}$
        $stop\_unfold$ = true
        forall $r_B \in R_B$ do
            forall $r_H \in R_H$ do
                if $do\_unfold(a, r_H, r_B, P)$ then
                    $stop\_unfold$ = false
                    add $r_H$ and $unfold(a, r_H, r_B)$ to $P'$
                    if $|\{b \in H(r_H)$ such that $b$ unifies with $a\}| > 1$ then
                        add $r_B$ to $P'$
                    end
                else
                    add $r_H$ and $r_B$ to $P'$
                end
            end
        end
        $P = P' \cup (P \setminus (R_B \cup R_H))$
    end
until $P^l = P$ or $stop\_unfold$ is true
return $filter(P, F)$
```

Algorithm 4: *RuleOptimizer*(P): Optimize the bodies of all cq-rules in P

```
Input: Set of cq-rules $P$
Result: pushed and variable eliminated cq-rules
foreach $r \in P$ such that $B^+(r) \neq \emptyset$ do
    choose $b \in B^+(r)$
    $B^+(r) = BodyOptimizer(b, B^+(r) \setminus \{b\}, \emptyset, \emptyset)$
    forall $a = \mathrm{DL}[\lambda; cq](\vec{Y})$ in $B^+(r)$ s.t. $X = t$ in $cq$ or $B^+(r)$ do
        if $X \notin \vec{Y}$ then
            $r = H(r) \leftarrow \mathrm{DL}\big[\lambda; cq_{X/t}\big](\vec{Y}), B^+(r) \setminus \{a\}, \mathrm{not}\, B^-(r)$
        else
            $r = H(r)_{X/t} \leftarrow \mathrm{DL}\big[\lambda; cq_{X/t}\big](\vec{Y} \setminus \{X\} \cup \omega(t)),$
                                  $(B^+(r) \setminus \{X = t, a\})_{X/t}, \mathrm{not}\, B^-(r)_{X/t}$
        end
    end
end
return $P$
```

Algorithm 5: *BodyOptimizer*(o, B, C, O): Push queries in body B wrt. o

```
Input: atom $o$, body $B$, carry $C$, and optimized body $O$
Result: pushed optimized body $B$
if $B \neq \emptyset$ then
    choose $b \in B$
    if $do\_push(o, b)$ then  $o = push(o, b)$ else  $C = C \cup \{b\}$
    if $|B| > 1$ then
        return $BodyOptimizer(o, B \setminus \{b\}, C, O)$
    else if $|C| \neq \emptyset$ then
        choose $c \in C$, return $BodyOptimizer(c, C \setminus \{c\}, \emptyset, O \cup \{o\})$
    end
end
return $O \cup \{o\}$
```

Example 5.10. Let us reconsider the region program on the wine ontology in Example 3.7. Using the optimization methods for cq-programs we obtain from P an equivalent program P', where we replace rule (1) and rule (5) in P by

$$visit(L) \lor \neg visit(L) \leftarrow \mathrm{DL}\left[\begin{array}{c} WhiteWine(W_1), RedWine(W_2), \\ locatedIn(W_1, L), locatedIn(W_2, L) \end{array}\right](W_1, W_2, L), \quad (1')$$
$$\mathrm{not}\,\mathrm{DL}\big[locatedIn(L, L')\big](L),$$

and

$$delicate\_region(W) \leftarrow visit(L), \mathrm{DL}\left[\begin{array}{c} hasFlavor(W, wine{:}Delicate), \\ locatedIn(W, L) \end{array}\right](W, L), \quad (5')$$

respectively. The dl-queries in rule (1) had been pushed into a single CQ; the result is (1'). Furthermore, to obtain (5'), rule (6) has been folded into rule (5), and subsequently Query Pushing was applied to it.

5.2.3 Cost-based query pushing

The functions *do_unfold* and *do_push* in Algorithms 3 and 5 determine whether we can benefit from unfolding or query pushing. Given the input parameters, they should know whether doing the operation leads to a "better" program in terms of evaluation time, size of the program, arity of cq- or ucq-atoms, data transmission time, etc.

In the database area, cost estimations are based on a cost model, which usually has information about the size of a database and its relations, an estimate of the selectivity of joins and selections, the cost of the data transfer, etc. In our setting, similar knowledge can be used to determine the cost for pushed queries.

Example 5.11. Suppose we have a description logic knowledge base L with concepts *Mammal* and *SeaAnimal*. Let $|Mammal| = m$ and $|SeaAnimal| = n$. Consider the rule

$$r : a(X) \leftarrow \mathrm{DL}[Mammal](X), \mathrm{DL}[SeaAnimal](X).$$

Without optimization we need to send to the DL-engine two queries and would get one answer with m mammals and another answer with n animals living in the sea. Usually, the intersection of *Mammal* and *SeaAnimal* is very small and we need to throw away a lot of unnecessary individuals.

Applying query pushing, we rewrite r to

$$r' : a(X) \leftarrow \mathrm{DL}[\mathit{Mammal}(X), \mathit{SeaAnimal}(X)](X).$$

and get all sea animals which are also mammals in only one step, thus saving one query and retrieve only a small amount of individuals in contrast to $m + n$ individuals.

Example 5.12. Consider the DL-KB L in Example 5.11, but now we want to compute the cartesian product of all sea animals and mammals. The corresponding rule is

$$r : a(X, Y) \leftarrow \mathrm{DL}[\mathit{Mammal}](X), \mathrm{DL}[\mathit{SeaAnimal}](Y).$$

Due to query pushing, we would obtain r' from r

$$r' : a(X, Y) \leftarrow \mathrm{DL}[\mathit{Mammal}(X), \mathit{SeaAnimal}(Y)](X, Y).$$

In this example, it is not obvious if we should push both dl-atoms to one cq-atom. Suppose $m = n = 10$, so in rule r we would transfer 20 tuples from the DL-engine and perform the join operation locally in the ASP solver. Contrary, in r' we would receive 100 tuples, which may cost more than two consecutive queries due to network latency.

Example 5.13. In this example we push a role query $\mathit{livesIn}(X, Y)$ to the concept query $\mathit{SmallTown}(Y)$. In rule

$$r : a(X, Y) \leftarrow \mathrm{DL}[\mathit{livesIn}](X, Y), \mathrm{DL}[\mathit{SmallTown}](Y),$$

we retrieve both extensions of *livesIn* and *SmallTown*, while in

$$r' : a(X, Y) \leftarrow \mathrm{DL}[\mathit{livesIn}(X, Y), \mathit{SmallTown}(Y)](X, Y),$$

we retrieve much less tuples, whenever *SmallTown* is small.

The previous examples may give some hints on optimization strategies based on the size of concept and role extensions. Another useful strategy is to exploit functional properties in OWL. An OWL property R is *functional*, if for all individuals x, y_1, y_2 it holds that $R(x, y_1) \wedge R(x, y_2) \rightarrow y_1 = y_2$, i.e., x is a key in R. Similarly, R is *inverse functional*, if R^- is functional, i.e., for all individuals x_1, x_2, y $R(x_1, y) \wedge R(x_2, y) \rightarrow x_1 = x_2$.

Example 5.14. That every person has only one mother may be stated by the functional property *hasMother*, which is expressed by the axiom $\top \sqsubseteq \leq 1.\mathit{hasMother}$. The following rule retrieves all mothers of men:

$$r : a(Y) \leftarrow \mathrm{DL}[\mathit{hasMother}](X, Y), \mathrm{DL}[\mathit{Man}](X).$$

After application of Query Pushing, we obtain the rule

$$r' : a(Y) \leftarrow \mathrm{DL}[\mathit{hasMother}(X, Y), \mathit{Man}(X)](X, Y).$$

In r we get two answers with size $|\mathit{hasMother}| + |\mathit{Man}|$, while in r' we retrieve at most $|\mathit{Man}|$ tuples due to the functional nature of *hasMother*. Pushing would be even more attractive if the concept used would be very selective, e.g., if we used *Nobel_Laureate* instead of *Man*.

Optimization techniques using a sophisticated cost model have been considered for answering single CQs in [Sirin and Parsia, 2006]. In the previously mentioned paper, this model has been shown to be very effective for optimizing conjunctive queries. The difference to our approach is that we know about all possible queries which occur in a cq-program in advance, hence we are able to draw a distinction between local queries in a single cq-rule and the effect of optimizing the whole program before the first query has been sent to the DL-reasoner. It is reasonable to take the optimization procedures of the DL-reasoners into account, owing to the fact that this would even more refine the effectiveness of optimized cq-programs.

5.3 Experimental Results

In this section, we provide experimental results for the rule transformations described in the above section and the performance gain obtained by applying the various optimization techniques. The results reinforce the optimization potential that are at the basis of our rewriting rules.

VICODI program: (Fact Pushing)

$$P_v = \left\{ \begin{array}{c} c(\mathit{vicodi{:}Economics}), c(\mathit{vicodi{:}Social}), \\ v(X) \leftarrow \mathrm{DL}[\mathit{hasCategory}](X,Y), c(Y). \end{array} \right\}$$

$$P'_v = \left\{ \begin{array}{l} c(\mathit{vicodi{:}Economics}), c(\mathit{vicodi{:}Social}), \\ v(X) \leftarrow \mathrm{DL}\left[\begin{array}{c} \left\{ \begin{array}{c} \mathit{hasCategory}(X,Y), \\ Y = \mathit{vicodi{:}Economics} \end{array} \right\} \vee \\ \left\{ \begin{array}{c} \mathit{hasCategory}(X,Y), \\ Y = \mathit{vicodi{:}Social} \end{array} \right\} \end{array} \right](X,Y). \end{array} \right\}$$

SEMINTEC query: (Query Pushing)

$$P_{s_2} = \left\{ \begin{array}{r} s_2(X,Y,Z) \leftarrow \mathrm{DL}[\mathit{Man}](X), \mathrm{DL}[\mathit{isCreditCard}](Y,X), \mathrm{DL}[\mathit{Gold}](Y), \\ \mathrm{DL}[\mathit{livesIn}](X,Z), \mathrm{DL}[\mathit{Region}](Z) \end{array} \right\}$$

$$P'_{s_2} = \left\{ s_2(X,Y,Z) \leftarrow \mathrm{DL}\left[\begin{array}{c} \mathit{Man}(X), \mathit{Gold}(Y), \mathit{Region}(Z), \\ \mathit{isCreditCard}(Y,X), \mathit{livesIn}(X,Z) \end{array} \right](X,Y,Z). \right\}$$

SEMINTEC costs: (Query Pushing, Functional Property)

$$P_l = \{ l(X,Y) \leftarrow \mathrm{DL}[\mathit{hasLoan}](X,Y), \mathrm{DL}[\mathit{Finished}](Y). \}$$
$$P'_l = \{ l(X,Y) \leftarrow \mathrm{DL}[\mathit{hasLoan}(X,Y), \mathit{Finished}(Y)](X,Y). \}$$

hasLoan is an inverse functional property and $|\mathit{hasLoan}| = 682(n+1)$, $|\mathit{Finished}| = 234(n+1)$, where n is obtained from the ontology instance SEMINTEC_n.

LUBM faculty: (Query Pushing, Inequality Pushing, Variable Elimination)

$$P_f = \left\{ \begin{array}{r} f(X,Y) \leftarrow \mathrm{DL}[\mathit{Faculty}](X), \mathrm{DL}[\mathit{Faculty}](Y), D_1 = D_2, U_1 \neq U_2, \\ \mathrm{DL}[\mathit{doctoralDegreeFrom}](X,U_1), \mathrm{DL}[\mathit{worksFor}](X,D_1), \\ \mathrm{DL}[\mathit{doctoralDegreeFrom}](Y,U_2), \mathrm{DL}[\mathit{worksFor}](Y,D_2). \end{array} \right\}$$

$$P'_f = \left\{ f(X,Y) \leftarrow \mathrm{DL}\left[\begin{array}{c} \mathit{Faculty}(X), \mathit{Faculty}(Y), U_1 \neq U_2, \\ \mathit{worksFor}(X,D_1), \mathit{worksFor}(Y,D_1), \\ \mathit{doctoralDegreeFrom}(X,U_1), \\ \mathit{doctoralDegreeFrom}(Y,U_2) \end{array} \right](X,Y,U_1,U_2,D_1). \right\}$$

Table 5.2: Some test queries

We have tested the rule transformations using the prototype implementation of the dl-plugin for dlvhex,[1] a logic programming engine featuring higher-order syntax and external atoms (see [Eiter et al., 2005b] and previous sections), which uses RacerPro 1.9 as DL-reasoner (cf. [Haarslev and Möller, 2001]). Note that RacerPro does not have full support for (U)CQs over DLs, instead, only ground (U)CQs are available, i.e., during the

[1]`http://www.kr.tuwien.ac.at/research/dlvhex/`

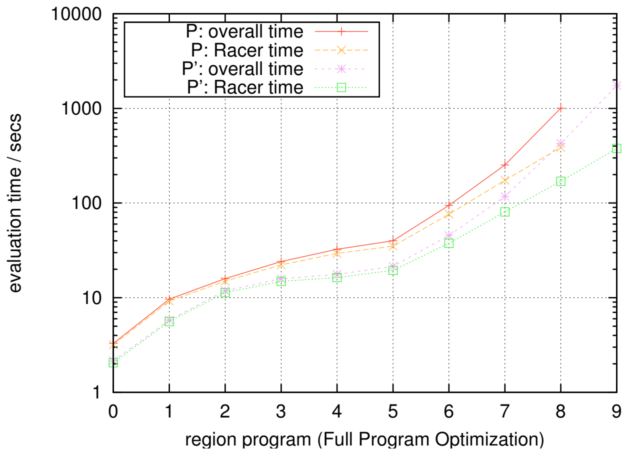

Figure 5.1: Evaluation time for region example

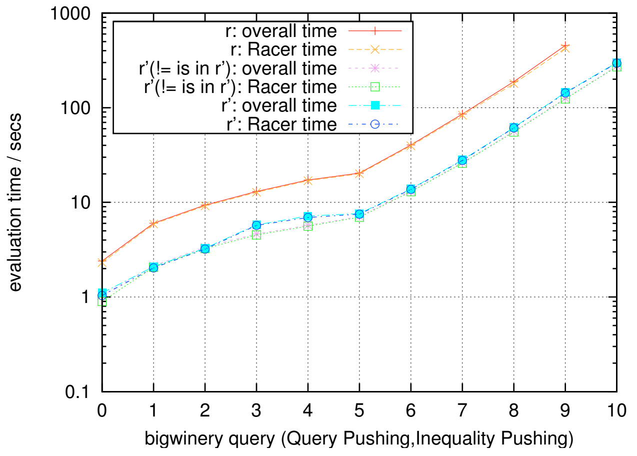

Figure 5.2: Evaluation time for bigwinery example

query evaluation process the non-distinguished variables in a given (U)CQ is replaced by known individuals from the DL-KB. As a consequence of this we cannot derive all consequences in cq-program P' in Example 3.2 with our dl-plugin. Current implementations of other DL-reasoners do not support the full range of (U)CQs either: the KAON2 engine[2] provides ground conjunctive queries like RacerPro, whereas the Pellet reasoner[3] fully adopts conjunctive queries with existential variables as long they do not contain cycles in the query graph. Advancements in the respective implementations of the DL-reasoners will close this gap, but due to our software architecture which advocates separation of the rule and ontology component, this would not affect our part of the implementation. To our knowledge, this is currently the only implemented system for such a coupling of nonmonotonic logic programs and description logics.

The tests were done on a P4 3GHz PC with 1GB RAM under Linux 2.6. As an ontology benchmark, we used the testsuite described in [Motik and Sattler, 2006].[4] The experiments covered the region program (Example 3.7) and its optimized version (Example 5.10), the bigwinery program in Example 5.3, as well as particular query rewritings, including the

[2]`http://kaon2.semanticweb.org/`
[3]`http://pellet.owldl.com/`
[4]Available at `http://kaon2.semanticweb.org/download/test_ontologies.zip`

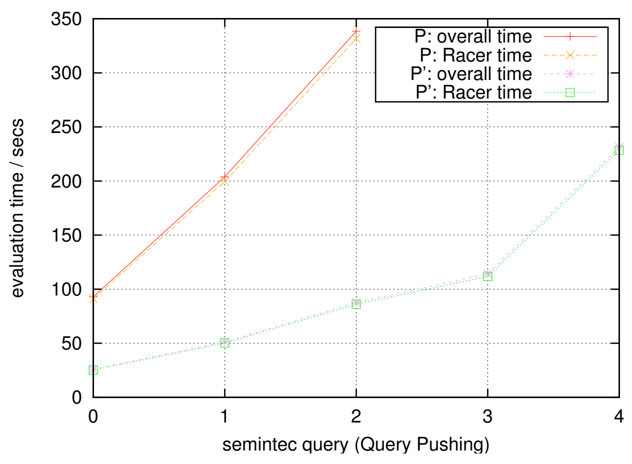

Figure 5.3: Evaluation time for SEMINTEC example

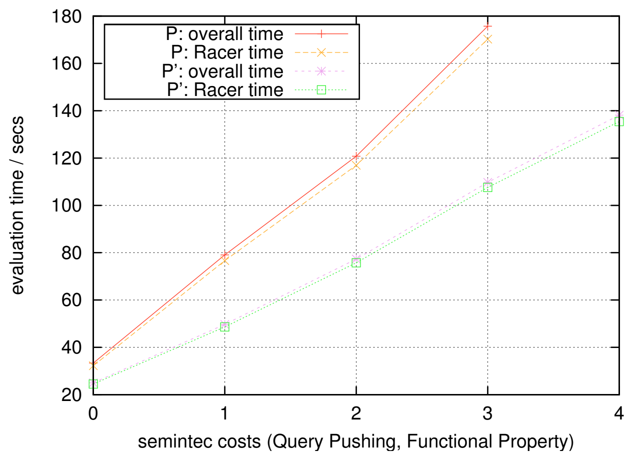

Figure 5.4: Evaluation time for SEMINTEC cost example

ones in Table 5.2 (P_{s_2} is taken from [Motik and Sattler, 2006]). The rewritten programs are presented below each experimental query in this table. The results are shown in Figures 5.1–5.6 and will be briefly explained in the following. The horizontal axis in these diagrams shows the used ontologies, and the vertical axis serves to display the used time in seconds (Figure 5.1 and 5.2 use a logarithmic scale). Missing entries in the diagrams indicate that the evaluation failed due to memory exhaustion during query evaluation of RacerPro. For fully detailed test results see Table B.1–B.6 in Appendix B. Statistical information of the used ontologies is listed in [Motik and Sattler, 2006] for the wine, SEMINTEC, and VICODI ontologies; the modified LUBM ontology is recorded in Table B.7.

In most of the tested programs, the performance boost using the aforementioned optimization techniques was substantial. Due to the size of the respective ontologies, in some circumstances the DL-engine failed to evaluate the original dl-queries, while the optimized programs did terminate with the correct result.

- In detail, for the region program, we used the ontologies wine_0 through wine_9. As can be seen from the graph in Figure 5.1, there is a significant speedup, and in case of wine_9 only the optimized program could be evaluated. Most of the computation time was spent by RacerPro. We note that the result of the join in the first rule had only size linear in the number of top regions L; a higher performance gain may be expected for ontologies with

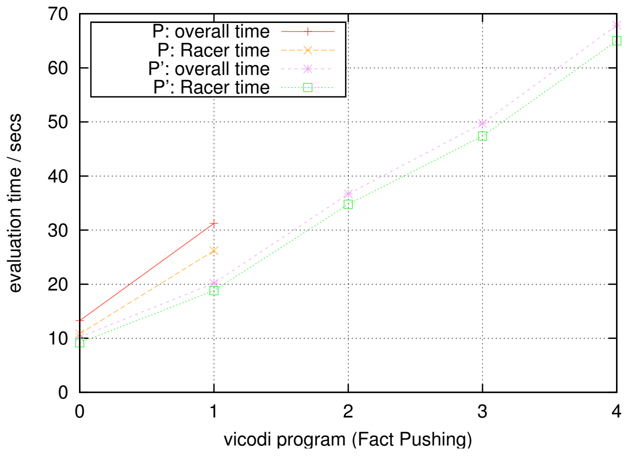

Figure 5.5: Evaluation time for VICODI example

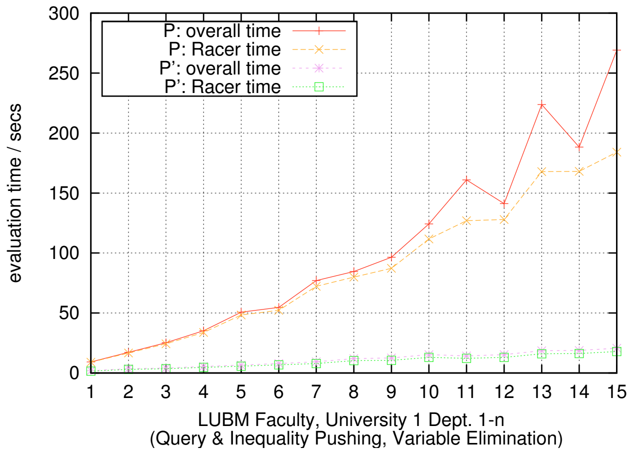

Figure 5.6: Evaluation time for LUBM faculty example

larger joins.

- We experimented with the bigwinery rule from Example 5.3 over various wine ontologies (see Figure 5.2). However, different technical realizations of inequalities (using `not` and `same-as` query atoms or injective variables) lead to largely diverging results. Using RacerPro's native inequality predicate (`not (same-as` $\cdots$)) resulted in a very bad runtime behaviour; in fact, issuing the bigwinery query with `same-as` atom in the nRQL expression lead to a much slower query evaluation on the RacerPro side. To overcome this flaw in the RacerPro implementation, we used so-called injective variables for W_1 and W_2, i.e., variables prefixed with a `?` sign in the nRQL query. Such variables are always bound to different individuals, therefore only different individuals show up in each tuple of the result and consequently are implicitly tied up through an inequality predicate. The test outcome shows that there is a huge advantage in using r' over r. Pushing the inequality inside the query part, however, increased the query evaluation time. Leaving the inequality in the cq-program, i.e., only apply Query Pushing to r, leads to the rule

$$r'' : \mathit{bigwinery}(M) \leftarrow \mathrm{DL}\left[\begin{array}{c} \mathit{Wine}(W_1),\ \mathit{Wine}(W_2), \\ \mathit{hasMaker}(W_1, M), \\ \mathit{hasMaker}(W_2, M) \end{array}\right](W_1, W_2, M), W_1 \neq W_2.$$

Using above rule gave a substantial better evaluation time for the bigwinery example. So it seems that there is a problem in RacerPro when it comes to inequality evalution in nRQL queries, because, as it turned out, the number of tuples transferred in case of r' with pushed inequality is 2.625 times smaller than in case of r''. From that point of view, inequalities inside of conjunctive queries should be preferred.

• The SEMINTEC tests dealt with Query Pushing for single rules. The rule in P_{s_2} is from one of the benchmark queries in [Motik and Sattler, 2006], while P_l tests the performance increase when pushing a query to a functional property (see Table 5.2). In both situations, we performed the tests on the ontologies SEMINTEC_0 up to SEMINTEC_4. As shown in Figure 5.3 and Figure 5.4 the evaluation speedup was significant. We could complete the evaluations of P_{s_2} on all SEMINTEC ontologies only with the optimization. The performance gain for P_l is in line with the constant join selectivity.

• The VICODI test series revealed the power of Fact Pushing (see Figure 5.5). While the unoptimized VICODI program (Table 5.2) could be evaluated only with ontologies VICODI_0 and VICODI_1, all ontologies VICODI_0 up to VICODI_4 could be handled with the optimized program.

• In the LUBM test setup, we used the LUBM Data Generator[5] to create the Department ontologies for University 1 (cf. [Guo et al., 2005]). We then created 15 ontologies out of this setup, where each ontology LUBM_n has Department 1 up to Department n in the ABox. See also Table B.7 for statistical information for our generated ontologies. The test query P_f results in Figure 5.6 show a drastic performance improvement.

[5] `http://swat.cse.lehigh.edu/projects/lubm/`

The goal of computer science is to build something that will last at least until we've finished building it.

—anonymous

6 Applications

In this chapter, we present a selection of applications for cq-programs. While the applicability of rule languages in the context of Semantic Web reasoning is evident, we will demonstrate two practical usage scenarios for our cq-programs in not so obvious fashions. The first application is set in the bioinformatics area, namely in the integration of different kinds of biomedical ontologies, and the second application for cq-programs deals with musical genre classification using ontologies and rules.

6.1 Biomedical Ontologies with Rules

Ontologies are very popular in the Life Sciences, especially biomedical ontologies in the bioinformatics community. As reported in [Hoehndorf et al., 2007], ontologies of that kind may be classified into two broad categories: *canonical* and *pathological* ontologies. The first range of ontologies are used to describe an idealized view of a biomedical domain such as anatomical relationships in a healthy human being. For instance, such ontologies are used to catalogue statements like "every human has an appendix." The second type of ontologies describes exceptional associations from the idealized view of canonical ontologies. Considering the general appendix statement, in most cases of appendicitis it is mandatory to remove the appendix, hence some human individuals live without this body part. Pathological ontologies are thus capable of modelling human beings which lack an appendix, whereas canonical ontologies would not be able to perform this task. Moreover, a human without appendix would introduce an inconsistency in canonical ontologies.

Now given that the combination of instances of these kinds of biomedical ontologies immediately lead to inconsistency, the authors of [Hoehndorf et al., 2007] argue that integration endeavours for creating a common ontological framework—which comprises of canonical and pathological ontologies—should be performed using a nonmonotonic reasoning machinery. Hoehndorf et al. show this by using the GFO-Bio ontology [Herre et al., 2006], which comes in form of an OWL file.[1] GFO-Bio is a *top-level ontology*, i.e., an ontology tailored for describing abstract categories and relationships which is suitable throughout multiple domains. In case of biomedical ontologies, top-level categories embrace concepts such as *Cell*, *Organism*, *Anatomical Part*, *Biological Process*, and similar biological subject matters. Top-level ontologies thus are usable to model domain-specific knowledge

[1] `http://www.onto-med.de/ontologies/gfo.owl`

in *domain ontologies*, which are restricted to a specific domain, for instance genetic or human anatomy ontologies.

In order to build a combined canonical and pathological domain ontology using the top-level concepts and roles in GFO-Bio, some form of nonmonotonic reasoning must be applied to circumvent inconsistencies similar to that in the appendix example earlier. Hoehndorf et al. do this by using *Default Logic* [Reiter, 1980]. Without going into details, a default logic theory comprises of a set of first-order sentences and a set of default rules, which can be seen as nonstandard inference rules. The next example shows a formalization of our running example of a prototypical human in default logic. In the light of our appendix example, a "default" human includes an appendix:

$$\frac{\mathit{human}(x) : \mathit{has\_part}(x, \mathit{appendix})}{\mathit{has\_part}(x, \mathit{appendix})} \tag{6.1}$$

The prior default rule should be read as "given a human x, if it is consistent to assume that x has an appendix, then we derive that x has an appendix." With such kind of nonmonotonic reasoning, the integration of canonical and pathological ontologies using GFO-Bio as top-level ontology is now doable; the consistency of the theory is preserved, while the inconsistency that would have been arisen from human beings without appendices is circumvented.

As shown in [Eiter et al., 2007b], dl-programs (and thus HEX-programs) are applicable for default reasoning with description logics. [Hoehndorf et al., 2007], too, considers such a setup for their implementation by using the above line of reasoning over integrated top-level ontologies. Hence, HEX-programs with DL external atoms come as a handy tool to support reasoning in such an ontological integration effort with built-in optimization features.

Example 6.1. An example for a HEX-program using GFO-Bio as upper ontology is the following variant of a HEX-program shown in [Hoehndorf et al., 2007]:

$$\begin{aligned}
\mathit{class}(X) &\leftarrow \&\mathit{dlC}[\text{“}\mathit{gfo\text{-}bio.owl}\text{”}, a, b, c, d, \text{“}\mathit{Category}\text{”}](X). && (1)\\
\mathit{ind}(X) &\leftarrow \&\mathit{dlC}[\text{“}\mathit{gfo\text{-}bio.owl}\text{”}, a, b, c, d, \text{“}\mathit{Individual}\text{”}](X). && (2)\\
\mathit{inst}(X, Y) &\leftarrow \&\mathit{dlR}[\text{“}\mathit{gfo\text{-}bio.owl}\text{”}, a, b, c, d, \text{“}\mathit{instance\_of}\text{”}]\,(X, Y), && (3)\\
&\qquad \mathit{ind}(X), \mathit{class}(Y).\\
\mathit{has\_part}(X, \mathit{appendix}) &\leftarrow \mathit{inst}(X, Y), && (4)\\
&\qquad \text{not } \&\mathit{dlR}[\text{“}\mathit{patho.owl}\text{”}, a, b, c, d, \text{“}\mathit{lacks\_part}\text{”}]\,(X, \mathit{appendix}),\\
&\qquad \&\mathit{dlR}[\text{“}\mathit{canonical.owl}\text{”}, a, b, c, d, \text{“}\mathit{has\_part}\text{”}]\,(Y, \mathit{appendix}).
\end{aligned}$$

In rule (1), we import all *Category* members into the predicate *class*. Similarly, (2) retrieves all *Individual* instances into *ind*. In (3), we get all X which are *instance_of* Y, where X is an *ind* and Y is a *class*. Rule (4) is a sophisticated version of default rule (6.1). In this rule, we integrate the pathological ontology *patho.owl* with the canonical ontology *canonical.owl* using the categories defined in *gfo-bio.owl*. This shows how nonmonotonic formalisms like HEX-programs with DL external atoms provide practical value in this kind of ontology integration.

Observe that in Example 6.1 various cq-program optimizations may be applied to support a more efficient program evaluation. For instance, various residual programs are obtained by unfolding the rules; after that, query pushing yields conjunctive queries to decrease the amount of queries to the GFO-Bio ontology.

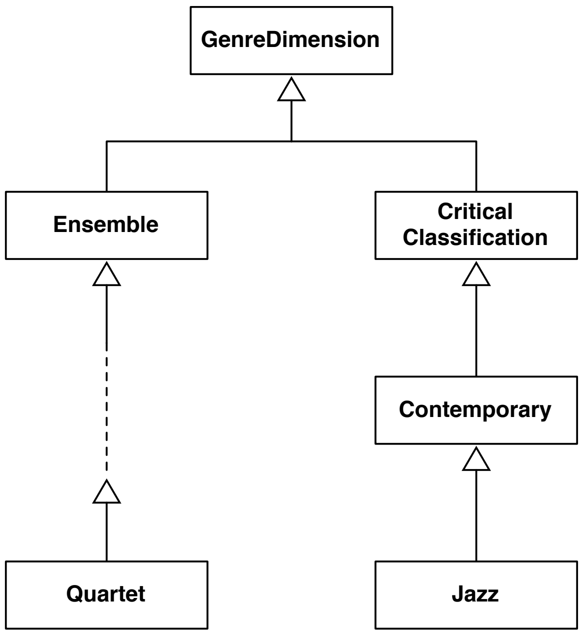

Figure 6.1: Genre Hierarchy in MX-Onto

6.2 Genre Classification with Ontologies and Rules

Genre Classification is a hot topic in the machine learning and music information retrieval community. Whole conferences are devoted to this subject. However, logical methods—usually orthogonal to the approaches in machine learning—using specifically tailored music ontologies may be applied to reason over musical genre relationships.

In [Ferrara et al., 2006], it is reported that rules are used to classify genres according to context-based representation of music. This representation is generated by score analysis, i.e., analysis of musical notation on score sheets. In this approach, a four-dimensional music resource context is built, which is made up of *Ensemble*, *Rhythm*, *Harmony*, and *Melody* features. The resource-context is then described in the MX-Onto ontology.[2] Basically, this ontology is composed of two parts: the *Context Layer*, which is composed of the extracted resource context information built from score analysis, and the *Genre Classification Layer*, which consists of a predefined genre taxonomy. The genre classification mechanism includes a set of SWRL rules [Horrocks et al., 2004], which, given the resource context information, infers the corresponding genre for a particular music resource.

Example 6.2. An example for a classification is the following SWRL rule, which classifies music resources into the *quartet* genre:

$$\begin{aligned} &\mathit{Music\_Resource}(?r) \wedge \mathit{ensemble}(?r, ?b) \wedge \mathit{number\_of\_parts}(?b, ?c) \wedge \\ &\quad \mathit{ensemble\_part}(?b, ?d) \wedge \mathit{performers}(?d, ?e) \wedge ?c = 4 \wedge ?e = 1 \Rightarrow \mathit{quartet}(?r). \end{aligned} \tag{6.2}$$

Informally, rule (6.2) classifies music resources $?r$ in an ensemble having exactly four parts performed by exactly one performer.

Figure 6.1 shows a small part of the concept hierarchy in MX-Onto. The *quartet* concept is a subclass of an *Ensemble*, whereas the more music genre specific concepts are subclasses from *CriticalClassification*.

[2]`http://islab.dico.unimi.it/ontologies/mxonto.owl`

The quality of the well-established methods for classifying music into genres—like machine learning algorithms—may be increased when using the aforementioned style of musical information processing. Some genres are typically hard to tell apart for the machine learning algorithms; jazz pieces are widely known specimen in the class of "hard genres" (see [Tzanetakis and Cook, 2002]).

We claim that the approach in [Ferrara et al., 2006] can be equally well performed in cq-programs. Take, for instance, our previous example and translate the SWRL rule (6.2) into a cq-program (L, P), where P is the cq-rule

$$q(R) \leftarrow \mathrm{DL}\left[\begin{array}{c} \mathit{Music\_Resource}(R), \mathit{ensemble}(R,B), \mathit{number\_of\_parts}(B,C), \\ \mathit{ensemble\_part}(B,D), \mathit{performers}(D,E), C=4, E=1 \end{array}\right](R)$$

and L is the context part of the MX-Onto ontology in `http://islab.dico.unimi.it/ontologies/mxonto-context.owl`. This newly created information for quartets can now be fed into the genre part of MX-Onto in the ontology `http://islab.dico.unimi.it/ontologies/mxonto-genre.owl` as L', using $\mathit{Quartet} \uplus q$ in all dl-atoms, and adding the answer set of P as part of the new classification program. For instance, retrieving all Jazz and Quartet exemplars from the ontology is made by using this rule in P':

$$\begin{array}{c} q(\mathit{quartet}_1),\ \ldots\ q(\mathit{quartet}_k), \\ \mathit{jazz\_quartet}(Q) \leftarrow \mathrm{DL}[\mathit{Quartet} \uplus q; \mathit{Jazz}(Q), \mathit{Quartet}(Q)](Q), \end{array}$$

where $\{q(\mathit{quartet}_1), \ldots, q(\mathit{quartet}_k)\}$ is the answer set of (L, P).

Since DL external atoms allow to specify the DL-KB as input, we describe the above mechanism as single HEX-program P', thus eliminating the intermediate step of adding the answer set of (L, P) to P':

$$\begin{array}{l} q(\text{“}\mathit{Quartet}\text{”}, R) \leftarrow \\ \&\mathit{dlCQ}\left[\text{“}\mathit{mxonto\text{-}context.owl}\text{”}, a, a, a, a, \begin{array}{r} \mathit{Music\_Resource}(R), \mathit{ensemble}(R,B), \\ \mathit{number\_of\_parts}(B,C), C=4, E=1, \\ \mathit{ensemble\_part}(B,D), \mathit{performers}(D,E) \end{array}\right](R). \\ \mathit{jazz\_quartet}(Q) \leftarrow \&\mathit{dlCQ}[\text{“}\mathit{mxonto\text{-}genre.owl}\text{”}, q, a, a, a, \text{“}\mathit{Jazz}(Q), \mathit{Quartet}(Q)\text{”}](Q). \end{array}$$

The extension of *jazz_quartet* contains all classified members of music resources for musical pieces falling into the jazz quartet genre. As one can easily imagine, nowadays statistical methods would probably fail to recognize such musical compositions without this kind of background knowledge.

If you wish to make an apple pie from scratch, you must first invent the universe.

—Carl Sagan, *Cosmos*

7 Conclusion

In this work, we have studied an extension to the dl-program [Eiter et al., 2004b, 2006a] formalism called cq-programs. dl-programs combine description logics knowledge bases and nonmonotonic logic programs. Built on this approach, our extension of cq-programs is composed of (i) allowing conjunctive queries (CQs) and unions of conjunctive queries (UCQs) over a Description Logic (DL) knowledge base, and (ii) extended disjunctive logic programs. The effect of this extension is a higher expressiveness in the logic programming part and in the queries to the DL part. cq-programs retain decidability of reasoning as long as answering CQs resp. UCQs over DLs is decidable.

As we have explored in this thesis, CQs and UCQs over DLs open the path for program optimization. By pushing CQs to the highly optimized DL-reasoner, significant speedups can be gained. Partial evaluation of logic programs can be exploited to bring particular queries closer together, which, eventually, are pushed to build a single and more efficient query. As the experimental results show, in some cases evaluation is only feasible in optimized programs. This implies that integrated formalisms like cq-programs must cope with optimization strategies.

We introduced the dl-plugin, which is an implementation for dl- and cq-programs—and more general, HEX-programs with DL external atoms—using the HEX-program solver dlvhex and DL-reasoner RacerPro. We showed that cq-programs without nonmontonic dl-atoms may be reformulated as HEX-programs and evaluated in dlvhex. Moreover, the dl-plugin hosts certain of the previously mentioned optimization methods for dl- and cq-programs as program rewriting module for HEX-programs, hence HEX-programs shall be optimized too. Query caching for ordinary dl-queries is realized in the DL external atoms; this is another valuable technique for reducing the total amount of calls to the DL-reasoner.

7.1 Future Work

The results are promising and suggest to further the path of optimization. To this end, refined strategies implementing the tests *do_unfold* and *do_push* in Algorithm 3 and 5 are desirable, as well as further rewriting rules for the query part and the logic program. In particular, an elaborated cost model for query answering, which is able to decide whether a rewriting action contributes positively to the evaluation speed, would be interesting. However, given the continuing improvements on DL-reasoners, such a model had to be revised more frequently. In general, results from the distributed database field should have potentiality to gain further insights. Another suggested extension would be to develop query caching for CQs. Remaining work includes also a thorough study on the complexity of the cq-program formalism.

In view of optimization techniques, syntactic rewriting and semantic caching provide valuable results, but the interaction between those two methods has not been investigated. It is possible that under certain conditions, caching might be more effective than rewriting the program, and vice versa; the aforementioned cost model could give even better results. At the time of writing, dlvhex does not support datalog queries. Once this feature is implemented, additional optimization paths like the magic-set rewriting method can be applied in our framework. Weight constraints in dl- and cq-programs, or HEX-programs with DL external atoms, have not been considered yet for program optimization. In addition, some features only present in HEX-programs like variable arguments in input lists of external atoms might result in novel rewriting rules; one idea in this direction is to adapt the Fact Pushing rewriting rule for covering this specific case.

As the current implementation supports RacerPro, another interesting issue is to interface with other DL-reasoners that host CQs, e.g., KAON2 or Pellet, and to compare the results. In particular, coupling with an engine for answering arbitrary CQs or UCQs on highly expressive DLs would be intriguing. Moreover, complexity results for conjunctive query answering in some tractable DLs show that ontologies expressed in these specific DLs may be translated into a relational database instance and CQ processing outsourced to a relational database system. This scenario is appealing for large ABox instances, since we would bypass the DL-engine and use standard database systems with support for huge databases and fast query answering. For this purpose, the dl-plugin has to interface with a relational database. Alternatively, the rewriting module of the dl-plugin could be adapted to use not yet available database external atoms.

Given that many proposed integrated ontology/rules formalisms exist (see Section 1.3), which are quite diverse and use different semantics, it would be absorbing to show that some of them could be automatically rewritten into an equivalent cq-program. Even if no full translation was possible, it would be interesting to look at specific restrictions of these languages and provide reformulation algorithms. This would have a big practical impact, since only some of the hybrid languages are implemented and virtually none of them deal with optimization issues. Additionally, this could provide helpful insights for combined knowledge base formalisms.

A Proofs

Is this the right room for an argument?

—Monty Python's Flying Circus, *Argument Clinic*

A.1 Proofs for Section 4

A.1.1 Proof Theorem 4.8

Proof. ($\Rightarrow$) Let I be a minimal model of KB. Towards a contradiction, suppose I_π is not a model of $\tau_L(P)$. There is an $r \in \mathit{grnd}(\tau_L(P))$ such that $I_\pi \not\models r$, which implies $I_\pi \not\models H(r)$ and $I_\pi \models B(r)$. This r must be a ground version for a rule in $\tau_L(\rho)$, where $\rho = a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m$ in P, such that w.l.o.g. $a = \mathrm{DL}[\lambda; q](\vec{X}) \in B(\rho)$ is the only dl-atom in ρ. For a ground substitution θ, we obtain two cases, (i) $r = \rho\theta$, or (ii) r is a ground version for a rule in $\pi_L(a)$.

(i) Either $I \models_L H(\rho\theta)$ and $I \models_L B(\rho\theta)$, or $I \not\models_L B(\rho\theta)$. For the first case, we immediately derive a contradiction for $I_\pi \not\models H(r)$, since $H(r) = H(\rho\theta)$ and $I \subseteq I_\pi$. In the second case, by Lemma 4.7, $I \not\models_L B(\rho\theta)$ implies $I_\pi \not\models B(r)$. Consequently, I_π is a model of $\tau_L(P)$.

(ii) Since I_π is the answer set of $\pi(P)$, it is a model of $\mathit{grnd}(\pi(P))$. Thus, we get a contradiction, since r is a ground version of a rule in $\pi_L(a)$ and $\pi_L(a) \subseteq \pi(P)$, hence $r \in \mathit{grnd}(\pi(P))$. As a result, I_π is a model of $\tau_L(P)$.

Towards another contradiction, suppose $J \subset I_\pi$ is a minimal model of $\tau_L(P)$. J is not a model of KB, there is a ground $r \in \mathit{ground}(P)$ such that $J \models_L B(r)$ and $J \not\models_L H(r)$. We obtain two cases, (i) $J \cap I \neq I$, i.e, J does not contain some literals from I, and (ii) $J \cap I = I$, i.e., J does not contain a ground atom with predicate from $\{pc_\lambda, mc_\lambda, pr_\lambda, mr_\lambda\}$. For case (i), $J \cap I \subset I$ and $I \subseteq I_\pi$. Since r is also in $\mathit{grnd}(\tau_L(P))$, by Lemma 4.7, $J \not\models r$. This is a contradiction to J being the minimal model of $\tau_L(P)$. For case (ii), we get a contradiction right off for J being not a model of KB, since I is a model for KB. Therefore, I_π is a minimal model of $\tau_L(P)$.

($\Leftarrow$) Assume I_π is a minimal model of $\tau_L(P)$. Towards a contradiction, suppose I is not a model of KB. There is an $r \in \mathit{ground}(P)$ such that $I \not\models_L r$, thus $I \not\models_L H(r)$ and $I \models_L B(r)$. Since τ_L does not change ordinary cq-rules, w.l.o.g. assume that r contains one dl-atom $a = \mathrm{DL}[\lambda; q](\vec{c})$ in $B(r)$. Now let $r' = \tau_L(r)$ with $a' = \&dlT[\text{“}L\text{”}, pc_\lambda, mc_\lambda, pr_\lambda, mr_\lambda, q](\vec{c}) \in B(r')$ and the corresponding $\&dlT$. From $I \models_L B(r)$ and Lemma 4.7, we get $I_\pi \models B(r')$, since $I \models_L a$ iff $I_\pi \models a'$. Since I_π is a model of r', we obtain $I_\pi \models H(r')$. But this is a contradiction to $I \not\models_L H(r)$, since $H(r') = H(r)$. Thus, I is a model of KB.

Towards another contradiction, suppose $J \subset I$ is a minimal model of KB. Let J_π be the answer set of $J \cup \pi(P)$, hence $J_\pi \subset I_\pi$. J_π is not a model of $\tau_L(P)$, thus there is an $r \in grnd(\tau_L(P))$ such that $J_\pi \models B(r)$ and $J_\pi \not\models H(r)$. Let $\rho \in P$ be a rule of form $a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m$, such that w.l.o.g. $a = \mathrm{DL}[\lambda; q](\vec{X}) \in B(\rho)$ is the only dl-atom in ρ. For a ground substitution θ, we obtain two cases, (i) $r = (a_1 \vee \cdots \vee a_k \leftarrow \tau_L(b_1), \ldots, \tau_L(b_m))\theta$, or (ii) r is a ground version of a rule in $\pi_L(a)$.

(i) Since $\rho\theta \in ground(P)$ and $J \models_L \rho\theta$, by Lemma 4.7, we get that J_π is a model for r. This is a contradiction for the assumption that $J_\pi \not\models r$.

(ii) By J_π being a model for $\pi(P)$ and $\pi(P) \subseteq \tau_L(P)$, we derive a contradiction, since $r \in grnd(\pi(P))$ and by assumption $J_\pi \not\models r$.

Therefore, I is a minimal model of KB. □

A.1.2 Proof Theorem 4.9

Proof. Let I be a strong answer set of KB and I_π be the answer set of $I \cup \pi(P)$. Let $r \in ground(P)$. I is a minimal model of the positive sP_L^I. Let ρ be the single rule $\tau_L(r) \setminus \pi_L(r)$. We distinguish the cases

(i) $r \in sP_L^I$: in this case, $I \models_L a$ for all $a \in B^+(r)$ and $I \not\models_L l$ for all $l \in B^-(r)$. In particular, all dl-atoms are satisfied by I. By Lemma 4.7, $I_\pi \models b$ for all $b \in B^+(\rho)$ and $I_\pi \not\models l$ for all $l \in B^-(\rho)$, hence $I_\pi \models B(\rho)$ and $\rho \in f\tau_L(P)^{I_\pi}$.

(ii) $r \notin sP_L^I$: here, $I \not\models_L a$ for an $a \in B^+(r)$ or $I \models_L l$ for an $l \in B^-(r)$. By Lemma 4.7, we obtain $I \not\models B(\rho)$, hence $\rho \notin f\tau_L(P)^{I_\pi}$.

Therefore, by Theorem 4.8, I_π is a minimal model of $f\tau_L(P)^{I_\pi}$ iff I is a minimal model of sP_L^I. Hence, I_π is an answer set of $\tau_L(P)$ iff I is a strong answer set of KB. □

A.2 Proofs for Section 5

In order to prove the main results, we first state some Lemmas.

Lemma A.1. Let $\mathrm{DL}[\lambda_1; cq_1](\vec{Y_1})$ and $\mathrm{DL}[\lambda_2; cq_2](\vec{Y_2})$ be two cq-atoms such that $\lambda_1 \doteq \lambda_2$, and θ be a ground substitution over domain $\vec{Y_1} \cup \vec{Y_2}$. Then, $I \models_L \mathrm{DL}[\lambda_1; cq_1](\vec{Y_1}\theta)$ and $I \models_L \mathrm{DL}[\lambda_2; cq_2](\vec{Y_2}\theta)$ iff $I \models_L \mathrm{DL}[\lambda_1; cq_1' \cup cq_2']((\vec{Y_1} \cup \vec{Y_2})\theta)$.

Proof. ($\Rightarrow$) Suppose $I \models_L \mathrm{DL}[\lambda_1; cq_1](\vec{Y_1}\theta)$ and $I \models_L \mathrm{DL}[\lambda_2; cq_2](\vec{Y_2}\theta)$. Therefore both $L \cup \lambda_1(I) \models \phi_{cq_1}(\vec{Y_1}\theta)$ and $L \cup \lambda_2(I) \models \phi_{cq_2}(\vec{Y_2}\theta)$ hold. Thus, $L \cup \lambda_1(I) \models \phi_{cq_1'}(\vec{Y_1}\theta) \wedge \phi_{cq_2'}(\vec{Y_2}\theta)$ because of $\lambda_1 \doteq \lambda_2$, and this implies that $L \cup \lambda_1(I) \models \phi_{cq_1' \cup cq_2'}((\vec{Y_1} \cup \vec{Y_2})\theta)$, because for the rewritten non-distinguished variables $\vec{Y_1'} \cup \vec{Y_2'}$ of $cq_1'(\vec{Y_1}\theta) \cup cq_2'(\vec{Y_2}\theta)$, it holds that $\vec{Y_1'} \cap \vec{Y_2'} = \emptyset$ due to the variable renaming during the rewriting. Consequently, $I \models_L \mathrm{DL}[\lambda_1; cq_1' \cup cq_2']((\vec{Y_1} \cup \vec{Y_2})\theta)$.

($\Leftarrow$) Let $I \models_L \mathrm{DL}[\lambda_1; cq_1' \cup cq_2']((\vec{Y_1} \cup \vec{Y_2})\theta)$, hence $L \cup \lambda_1(I) \models \phi_{cq_1' \cup cq_2'}((\vec{Y_1} \cup \vec{Y_2})\theta)$ implies that both $L \cup \lambda_1(I) \models \phi_{cq_1}(\vec{Y_1}\theta)$ and $L \cup \lambda_1(I) \models \phi_{cq_2}(\vec{Y_2}\theta)$ hold. From $\lambda_1 \doteq \lambda_2$, we conclude that $L \cup \lambda_2(I) \models \phi_{cq_2}(\vec{Y_2}\theta)$, hence $I \models_L \mathrm{DL}[\lambda_1; cq_1](\vec{Y_1}\theta)$ and $I \models_L \mathrm{DL}[\lambda_2; cq_2](\vec{Y_2}\theta)$.

For the case were we have UCQs $ucq_1 = \bigvee_{i=1}^{r_1} cq_{1,i}$ and $ucq_2 = \bigvee_{i=1}^{r_2} cq_{2,i}$ in place of cq_1 and cq_2, respectively, the proof is straightforward. We just use $\bigvee_{i=1}^{r_1} \left(\bigvee_{j=1}^{r_2} cq_{1,i}' \cup cq_{2,j}' \right)$ instead of $cq_1' \cup cq_2'$. □

Lemma A.2. Let $a = \mathrm{DL}[\lambda_1; cq \cup \{X = t\}](\vec{Y})$ and $b = \mathrm{DL}[\lambda_2; cq_{X/t}](\vec{Y} \setminus \{X\} \cup \omega(t))$ be cq-atoms such that $\lambda_1 \doteq \lambda_2$, and θ be a ground substitution over domain $\vec{Y}$. The following statements hold:

(1) If $t \in \vec{Y}$ and L is under UNA, then $I \models_L a\theta$ iff $I \models_L b\theta$.

(2) If $t \notin \vec{Y}$, then $I \models_L a\theta$ iff $I \models_L b\theta$.

Proof. (1) ($\Rightarrow$) Suppose $I \models_L a\theta$. $(X = t)\theta$ and UNA in L implies that $X\theta$ and $t\theta$ denote the same individual symbol. Hence, $cq(\vec{Y}\theta) = cq_{X/t}(\vec{Y} \setminus \{X\} \cup \omega(t))\theta$ (even if X or t do not occur in the query atoms in cq) and $\lambda_1 \doteq \lambda_2$ implies $I \models_L b$.

($\Leftarrow$) Now suppose $I \models_L b\theta$. Since X does not appear in b, we replace occurrences of t in $cq_{X/t}(\vec{Y} \setminus \{X\} \cup \omega(t))$ to X such that $cq(\vec{Y})$ is obtained. Moreover, setting $X\theta$ to $t\theta$ implies $(X = t)\theta$. Therefore, $I \models_L a\theta$.

(2) The proof is essentially the same as (1). Here, we do not need UNA for replacing X by t, since t is not in the domain of θ. $X = t$ assures then that both terms denote the same individual in the universe. See also Lemma 6.1 in [Nonnengart and Weidenbach, 2001]. □

Lemma A.3. Let $\mathrm{DL}[\lambda_1; cq](\vec{Y}\theta)$ and $\mathrm{DL}[\lambda_2; cq \cup \{X \neq t\}](\vec{Y}\theta)$ be cq-atoms such that $\lambda_1 \doteq \lambda_2$, $X \in \vec{Y}$, and θ be a ground substitution over a domain $\vec{Y}$. Then, for L being under UNA, $I \models_L \mathrm{DL}[\lambda_1; cq](\vec{Y}\theta)$ and $I \models_L (X \neq t)\theta$ iff $I \models_L \mathrm{DL}[\lambda_2; cq \cup \{X \neq t\}](\vec{Y}\theta)$.

Proof. ($\Rightarrow$) Suppose $I \models_L \mathrm{DL}[\lambda_1; cq](\vec{Y}\theta)$ and $I \models_L (X \neq t)\theta$. We derive that $X\theta$ and $t\theta$ are syntactically different. Hence, $cq(\vec{Y}\theta) \cup \{X \neq t\}\theta$ holds in $L \cup \lambda_2(I)$ by $\lambda_1 \doteq \lambda_2$, therefore $I \models_L \mathrm{DL}[\lambda_2; cq \cup \{X \neq t\}](\vec{Y}\theta)$.

($\Leftarrow$) Now we assume that $I \models_L \mathrm{DL}[\lambda_2; cq \cup \{X \neq t\}](\vec{Y}\theta)$. Since $L \cup \lambda_2(I)$ satisfy $cq(\vec{Y}\theta)$ and $\{X \neq t\}\theta$, we conclude that $L \cup \lambda_1(I) \models_L cq(\vec{Y}\theta)$ and hence $I \models_L \mathrm{DL}[\lambda_1; cq](\vec{Y}\theta)$ and $I \models_L (X \neq t)\theta$. □

Lemma A.4. Let r and r' be positive cq-rules of form

$$r:\ H \leftarrow \mathrm{DL}\left[\lambda_1; \bigvee_{i=1}^{r} cq_i\right](\vec{Y}), f(\vec{Y'}), B$$

and

$$r':\ H \leftarrow \mathrm{DL}\left[\lambda_2; \bigvee_{i=1}^{r}\left(\bigvee_{j=1}^{l} cq_i \cup \left\{\vec{Y'} = \vec{c_j}\right\}\right)\right](\vec{Y}), B,$$

respectively, where $\lambda_1 \doteq \lambda_2$ and $\vec{Y'} \subseteq \vec{Y}$, θ be a ground substitution over a domain $\vec{Y}$, and I be a Herbrand interpretation such that $f(\vec{c_j}) \in I$ for $1 \leq j \leq l$ are all the literals with predicate f in I. Then, $I \models_L r\theta$ if and only if $I \models_L r'\theta$.

Proof. ($\Rightarrow$) Assume $I \models_L f(\vec{c_j})$ for $1 \leq j \leq l$ and $I \models_L r\theta$ hold. By $I \models_L r\theta$, either (i) $I \models_L H(r\theta)$ and $I \models_L B^+(r\theta)$ or (ii) $I \not\models_L B^+(r\theta)$.

(i) $I \models_L B^+(r\theta)$ implies $I \models_L f(\vec{Y'}\theta)$. Since $I \models_L f(\vec{c_j})$ for $1 \leq j \leq l$, we obtain that $f(\vec{Y'}\theta) = f(\vec{c})$ for a $\vec{c} \in \{\vec{c_1}, \ldots, \vec{c_l}\}$. Thus, the disjunction over $cq_i \cup \{\vec{Y'}\theta = \vec{c_j}\}$ for $1 \leq j \leq l$ must hold for some $\vec{c_j} = \vec{c}$. By $\lambda_1 \doteq \lambda_2$, we obtain $L \cup \lambda_2(I) \models \bigvee_{i=1}^{r}\left(\bigvee_{j=1}^{l} cq_i \cup \left\{\vec{Y'}\theta = \vec{c_j}\right\}\right)$, therefore I satisfies

$$\mathrm{DL}\left[\lambda_2; \bigvee_{i=1}^{r}\left(\bigvee_{j=1}^{l} cq_i \cup \left\{\vec{Y'}\theta = \vec{c_j}\right\}\right)\right](\vec{Y}\theta)$$

under L, and $I \models_L r'\theta$.

(ii) We obtain another two cases. First, $I \not\models_L B\theta$ implies $I \not\models B^+(r'\theta)$, hence $I \models_L r'\theta$. Secondly, some of $f(\vec{Y'}\theta)$ and $\mathrm{DL}[\lambda_1; \bigvee_{i=1}^{r} cq_i](\vec{Y}\theta)$ are not satisfied under L. By $\lambda_1 \doteq \lambda_2$, this implies that I does not satisfy $\mathrm{DL}\Big[\lambda_2; \bigvee_{i=1}^{r} \Big(\bigvee_{j=1}^{l} cq_i \cup \Big\{\vec{Y'}\theta = \vec{c_j}\Big\}\Big)\Big](\vec{Y}\theta)$ under L either, thus $I \models_L r'\theta$.

($\Leftarrow$) Assume $I \models_L r'\theta$. Either $I \models_L H(r'\theta)$ and $I \models_L B^+(r'\theta)$, or $I \not\models_L B^+(r'\theta)$. Similar to the ($\Rightarrow$) direction, we obtain now that $I \models_L r\theta$. □

The next Lemma is a generalization of Lemma 2.20 to cq-programs.

Lemma A.5. Let (L, P) be a positive cq-program and I a minimal model of (L, P). Then, an atom a is in I iff there is a ground rule $a \vee H \leftarrow B$ from P such that $I \setminus \{a\} \models B$ and $I \setminus \{a\} \not\models H$.

Proof. ($\Rightarrow$) Suppose for some atom a in I, there is no ground rule $a \vee H \leftarrow B$ from P such that $I \setminus \{a\} \models_L B$ and $I \setminus \{a\} \not\models_L H$. Then, for each ground rule r of the form $a \vee H \leftarrow B$, $I \setminus \{a\} \not\models_L B$ or $I \setminus \{a\} \models_L H$; hence it holds that $I \setminus \{a\} \models_L B$ implies $I \setminus \{a\} \models_L H$. In this case, $I \setminus \{a\}$ satisfies each rule r and becomes a model of (L, P), which contradicts the assumption that I is a minimal model. Hence the result follows.

($\Leftarrow$) Assume that a is not in I. Then $I \setminus \{a\} = I$, and for a ground rule $a \vee H \leftarrow B$ in P, $I \models_L B$ and $I \not\models_L H$ imply $a \in I$, which is a contradiction. □

A.2.1 Proof Theorem 5.5

In the following, let ρ be a rule of form r_1, r_2 (i.e., of form (B1a) resp. (B1b)), or r (i.e., (A1) resp. (C1)). Let r' be a rule of form (A2), (B2), and (C2), resp. Then, let $P' = (P \setminus \{\rho\}) \cup \{r'\}$, where ρ and r' are equivalent rules according to the rewriting rules (A), (B), or (C). We will show now that I is a (strong) answer set of (L, P) iff I is a (strong) answer set of (L, P').

Proof for (A), (B), and (C)

Proof. We first show for positive cq-programs (L, P), I is a minimal model of (L, P) iff I is a minimal model of (L, P').

($\Rightarrow$) Suppose I is a minimal model of (L, P). Towards a contradiction, assume I is not a model of (L, P'). Thus, for a ground substitution θ, there is a ground version of r' in $\mathit{ground}(P')$, $r'\theta$, such that $I \not\models_L H(r'\theta)$ and $I \models_L B(r'\theta)$. Since $I \models_L P$, in particular $\rho\theta \in \mathit{ground}(P)$, we get that (i) $I \models_L B(\rho\theta)$ and $I \models_L H(\rho\theta)$, or (ii) $I \not\models_L B(\rho\theta)$. In case of (i), we get a contradiction for $I \not\models_L H(r'\theta)$, since $I \models_L H(\rho\theta)$ and $H(\rho\theta) = H(r'\theta)$, hence I is a model of (L, P'). Now for case (ii), we have that $I \not\models_L B(\rho\theta)$, hence a literal of $B(\rho\theta)$ is false in I. If $a \in B(\rho\theta)$ is false in I, then $a \in B(r'\theta)$ is false in I by Lemma A.1 or A.3 (resp. A.2) for ρ of form r (resp. r_1 or r_2), which is a contradiction for $I \models_L B(r'\theta)$. Again, I is a model of (L, P').

Now assume that $J \subset I$ is a minimal model of (L, P'), therefore J is not a model of (L, P). For a ground substitution θ, there is a ground version of $\rho\theta$ in $\mathit{ground}(P)$ such that $J \not\models_L H(\rho\theta)$ and $J \models_L B(\rho\theta)$. Since $J \models_L r'\theta$ for a ground $r'\theta \in \mathit{ground}(P')$, we obtain the following cases. If $J \models_L B(r'\theta)$ and $J \models_L H(r'\theta)$, we derive a contradiction, since $H(\rho\theta) = H(r'\theta)$. Otherwise, if $J \not\models_L B(r'\theta)$, we derive a contradiction at $J \models_L B(\rho\theta)$, since Lemma A.1, A.2, and A.3 applies here as well. Consequently, I is a minimal model of (L, P').

($\Leftarrow$) Let I be a minimal model of (L, P'). We assume now that I is not a model of (L, P). Thus, for a ground substitution θ, there is a ground version of ρ in $\mathit{ground}(P)$, $\rho\theta$, such that $I \not\models_L H(\rho\theta)$ and $I \models_L B(\rho\theta)$. By $(\Rightarrow)$, we derive a contradiction, hence I is a model of (L, P).

To show that I is also a minimal model of (L, P), assume the contrary, there is a $J \subset I$ such that J is a minimal model of (L, P). This entails that J is not a model for P'. Again, using $(\Rightarrow)$ and Lemma A.1, A.2, or A.3, we conclude that J cannot be a minimal model of (L, P), hence I is a minimal model of (L, P).

Now we establish the proof for rewriting rules (A), (B), and (C) in Section 5.1.1, 5.1.2, and 5.1.3, respectively.

Let I be a strong answer set of (L, P). Since sP^I_L and sP'^I_L are positive cq-programs, we show now that I is a minimal model of sP^I_L iff I is a minimal model of sP'^I_L. To this end, consider a ground rule of form $\rho \in P$ with $\rho\theta \in \mathit{ground}(P)$, where θ is a ground substitution. We distinguish the cases:

(i) $\rho\theta \notin sP^I_L$: this implies that $I \not\models_L a$ for $a \in B^+(\rho\theta) \cap DL^?_P$, or $I \models_L l$ for $l \in B^-(\rho\theta)$. We conclude that $r'\theta \notin sP'^I_L$, since whenever $b \in B^+(\rho\theta) \cap DL^?_P$ is used in the process of the rewriting, and I does not satisfy b, and by the actual Lemma A.1, A.2, or A.3, I does not satisfy $b' \in B^+(r'\theta) \cap DL^?_{P'}$ either, where b' is the outcome of the resp. rewriting rule.

(ii) $\rho\theta \in sP^I_L$: then, $I \models_L a$ for all $a \in B^+(\rho\theta) \cap DL^?_P$, and $I \not\models_L l$ for all $l \in B^-(\rho\theta)$. Therefore, by applying the actual Lemma A.1, A.2, or A.3, $r'\theta \in sP'^I_L$.

Thus, $sP^I_L = sP'^I_L$, which implies I is a minimal model of sP^I_L iff I is a minimal model of sP'^I_L. Therefore, (L, P) has the same answer sets as (L, P'). □

A.2.2 Proofs Theorem 5.6 and 5.7

We split the proofs for Theorem 5.6 and 5.7 in two parts, the first part considers rewriting rule (D) of Theorem 5.6, while the second part deals with rewriting rules (E) and (F) of Theorem 5.6 and 5.7, respectively. We will show for each part of the proof that I is a strong answer set of (L, P) iff I is a strong answer set of (L, P').

Proof for (D)

Proof. We first show that for positive cq-programs $(L, \bar{P})$ and $(L, \bar{P}')$, the minimal models coincide. By Lemma A.4, we get for positive $\bar{P}' = (\bar{P} \setminus \{r\}) \cup \{r'\}$ a logically equivalent set of cq-rules, hence the minimal models of $(L, \bar{P})$ and $(L, \bar{P}')$ coincide.

Now let (L, P) and (L, P') be positive cq-programs, where $\bar{P} \subseteq P$ and $P' = (P \setminus \bar{P}) \cup \bar{P}'$ such that f does not occur in the heads of $P \setminus \bar{P}$. Since P' is logically equivalent to P, we obtain that the minimal models of (L, P) and (L, P') coincide.

For the general case, $(L, \bar{P})$ and $(L, \bar{P}')$ are cq-programs without restriction, we show now that $s\bar{P}^I_L = (s\bar{P}^I_L)'$, where $(s\bar{P}^I_L)'$ is obtained from applying rewriting rule (D) to the ground program $(L, s\bar{P}^I_L)$.

Let I be a strong answer set of $(L, \bar{P})$. I is a minimal model of the positive cq-program $(L, s\bar{P}^I_L)$. As shown above, I is a minimal model of $(L, s\bar{P}^I_L)$ iff I is a minimal model of $(L, (s\bar{P}^I_L)')$. Consider $r \in \bar{P}$, for a ground substitution θ of r; we obtain the case distinction:

(i) $I \not\models_L B^-(r\theta)$ and $I \models_L B^+(r\theta) \cap DL^?_{\bar{P}}$: In this case, $r\theta \in s\bar{P}^I_L$, therefore $r'\theta \in (s\bar{P}^I_L)'$. Since $r' \in \bar{P}'$ and (i) hold, we conclude that $r' \in s\bar{P}'^I_L$.

(ii) for some $l \in B^-(r\theta)$, $I \models_L l$, or for some $a \in B^+(r\theta) \cap DL^?_{\bar{P}}$, $I \models_L a$ hold: In this case, $r\theta \notin s\bar{P}^I_L$, therefore $r'\theta \notin (s\bar{P}^I_L)'$. Since $r' \in \bar{P}'$ and (ii) hold, we conclude that $r' \notin s\bar{P}'^I_L$.

Thus, $s\bar{P}'^I_L = (s\bar{P}^I_L)'$, that is, the reduct of the rewritten rules $\bar{P}'$ is equal to the rewritten rules of the reduct of $\bar{P}$, hence I is a minimal model of $(L, (s\bar{P}^I_L)')$ iff I is a minimal model of $(L, s\bar{P}'^I_L)$. Therefore, I is a strong answer set of $(L, \bar{P})$ iff I is a strong answer set of $(L, \bar{P}')$.

Now we are ready to finish the proof and show that for unrestricted (L, P) and (L, P'), where $\bar{P} \subseteq P$ and $P' = (P \setminus \bar{P}) \cup \bar{P}'$ such that f does not occur in the heads of $P \setminus \bar{P}$. Since P' is logically equivalent to P, we obtain that the strong answer sets of (L, P) and (L, P') coincide. □

Proof for (E) and (F)

Proof. We first show that for positive cq-programs $(L, \bar{P})$ and $(L, \bar{P}')$, the minimal models coincide.

To this end, let $\bar{P}$ consists of the positive cq-rules

$$r : H \leftarrow a(\vec{Y}), B$$

and

$$r_1 : H' \vee a(\vec{Y'}) \leftarrow B',$$

where $B = b_1, \ldots, b_m$, $B' = b'_1, \ldots, b'_n$, $H = a_1 \vee \cdots \vee a_k$, $H' = a'_1 \vee \cdots a'_l$, and $DL_{\bar{P}} = DL^+_{\bar{P}}$, such that for an mgu θ of $a(\vec{Y})$ and $a(\vec{Y'})$, $a(\vec{Y}\theta) = a(\vec{Y'}\theta)$. And let $\bar{P}'$ be consists of all the rules in $\bar{P}$ and the positive cq-rule

$$r'_1 : H'\theta \vee H\theta \leftarrow B'\theta, B\theta.$$

Due to the unfolding rule (E), $\bar{P}' = \bar{P} \cup \{r'_1\}$, which is logically equivalent to $\bar{P}$, hence $(L, \bar{P})$ and $(L, \bar{P}')$ have the same minimal models and thus $\bar{P} \equiv_L \bar{P}'$. Similarly, when $\bar{P} \subseteq P$ for an arbitrary positive set of cq-rules P and $P' = P \cup \{r'_1\}$, I is a minimal model of P iff I is a minimal model of P'.

Now we show that in case of Complete Unfolding (F), the positive cq-program (L, P) has the same minimal models as the positive cq-program (L, P').

Let r be as above, Q be a set of positive cq-rules such that no rules of form r and r_i appear in it, where r_i is a cq-rule of form

$$r_i : \; H_i \vee a(\vec{Y_i}) \leftarrow B_i \qquad (1 \leq i \leq l),$$

such that each H_i either does not contain a literal of form $a(\vec{Z})$, or no $a(\vec{Z}) \in H_i$ is unifiable with $a(\vec{Y})$; and P be the set of cq-rules $Q \cup \{r\} \cup \{r_i \mid 1 \leq i \leq l\}$, while $P' = (P \setminus \{r\}) \cup \{r'_i : H_i\theta_i \vee H\theta_i \leftarrow B_i\theta_i, B\theta_i \quad (1 \leq i \leq l)\}$ for mgus θ_i such that $a(\vec{Y})$ and $a(\vec{Y_i})$ unify.

($\Rightarrow$) Assume I is a minimal model of (L, P). Since I satisfies ground versions of r and all ground r_i, we obtain that I satisfies all of the corresponding ground versions of r'_i. Thus, we get that I is a model of (L, P'). Towards a contradiction, assume that J is a minimal model of (L, P'), such that $J \subset I$. J is not a model of (L, P) and a ground r must occur unsatisfied in *ground*(P), thus for a ground substitution η of r, $J \not\models_L r\eta$, which implies $J \models_L B(r\eta)$ and $J \not\models_L H(r\eta)$. By $J \models_L B(r\eta)$, it follows that $J \models_L a(\vec{Y}\eta)$. By Lemma A.5, we get for a ground $r'\sigma$ of a rule $r' \in P'$, i.e., either $r'_i\sigma$ or $r_i\sigma$, where σ is a ground substitution, $a(\vec{Y}\eta) \in H(r'\sigma)$. Since $r'\sigma = r'\sigma\eta$, we get $a(\vec{Y}\eta) = a(\vec{Y}\sigma\eta)$, and hence $a(\vec{Y}\sigma\eta) \in H(r'\sigma\eta)$. From $J \models_L B(r\eta)$ and $J \not\models_L H(r\eta)$, we conclude $J \models_L B(r\sigma\eta)$ and $J \not\models_L H(r\sigma\eta)$. We distinguish the cases:

(i) $r' = r'_i$: Assume that $r'\sigma\eta$ is a ground instance of r'_i and $a(\vec{Y}\sigma\eta) \in H(r'_i\sigma\eta)$. The mgu θ_i of $a(\vec{Y})$ and $a(\vec{Y_i})$ implies $\sigma\eta = \theta_i\rho$ for some ρ. Since $a(\vec{Y}\theta_i) = a(\vec{Y_i}\theta_i)$, we get $a(\vec{Y}\sigma\eta) = a(\vec{Y_i}\sigma\eta)$. Since $a(\vec{Y})$ does not occur unifiable in H_i of r_i, $a(\vec{Y_i}\sigma\eta) \notin H_i\sigma\eta$. Thus, $a(\vec{Y_i}\sigma\eta)$ must be one of $H\theta\sigma\eta$. $J \models_L a(\vec{Y}\sigma\eta)$ implies $J \models_L H\theta\sigma\eta$, but this contradicts $J \not\models_L H(r\sigma\eta)$. Therefore, I is also a minimal model of (L, P').

(ii) $r' = r_i$: Now suppose $r'\sigma\eta$ is a ground instance of r_i with $a(\vec{Y}\sigma\eta) = a(\vec{Y_i}\sigma\eta)$. Applying Lemma A.5, from $a(\vec{Y}\sigma\eta) \in J$, we conclude $J \setminus \{a(\vec{Y}\sigma\eta)\} \models_L B(r_i\sigma\eta)$ and $J \setminus \{a(\vec{Y}\sigma\eta)\} \not\models_L H_i\sigma\eta$. Since $a(\vec{Y}\sigma\eta) \notin B(r_i\sigma\eta)$, we get $J \models_L B(r_i\sigma\eta)$. Since $a(\vec{Y})$ does not occur unifiable in H_i of r_i, we obtain $a(\vec{Y}\sigma\eta) \notin H_i\sigma\eta$ and also $J \not\models_L H_i\sigma\eta$. $J \models_L B(r\sigma\eta)$ and $J \not\models_L H(r\sigma\eta)$ now implies that $J \not\models_L r'_i\sigma\eta$. Since $a(\vec{Y}\sigma\eta) = a(\vec{Y_i}\sigma\eta)$ and $a(\vec{Y}\theta_i) = a(\vec{Y_i}\theta_i)$, we get $\sigma\eta = \theta_i\rho$ for some ρ, thus $r'_i\sigma\eta$ is a ground instance of $r'_i \in P'$. Hence, $r'_i\rho \in \mathit{ground}(P')$ is not satisfied, which contradicts the assumption, that J is a model of (L, P'). Therefore, I is a minimal model of (L, P').

($\Leftarrow$) Let I be a minimal model of (L, P'). Assuming that I is not a model of (L, P), then $I \not\models_L r\eta$ for a ground substituion η. This implies $I \not\models_L H(r\eta)$ and $I \models_L B(r\eta)$, which in turn guarantees that $I \models_L a(\vec{Y}\eta)$. By Lemma A.5, we obtain for a ground version $r'\sigma$ of a rule $r' \in \bar{P}'$, i.e., either $r'_i\sigma$ or $r_i\sigma$, where σ is a ground substitution, $a(\vec{Y}\eta) \in H(r'\sigma)$. We will now apply a similar proof to the ($\Rightarrow$) direction and get the desired contradictions. Thus, I is a model of (L, P). Now we show that I is in fact a minimal model. To this end, assume that there is a minimal model $J \subset I$ of (L, P). Proceeding as in ($\Rightarrow$), J is also a minimal model of (L, P'), which contradicts our assumption that I is a minimal model of (L, P'), hence I is also a minimal model of (L, P).

Now we turn our attention to the general case, that is, (L, P) and (L, P') are cq-programs without restrictions. We show that $sP'^I_L = (sP^I_L)'$, where $(sP^I_L)'$ is the complete unfolded positive cq-program of the reduct of (L, P).

Let I be a strong answer set of (L, P). I is a minimal model of (L, sP^I_L), which is a positive program. Hence, by our first part of the proof, I is a minimal model of (L, sP^I_L) iff I is a minimal model of $(L, (sP^I_L)')$. Let us consider $r, r_i \in P$, we have an mgu θ_i for $a(\vec{Y}\theta_i) = a(\vec{Y_i}\theta_i)$, and for a ground substitution η, $a(\vec{Y}\eta) = a'(\vec{Y_i}\eta)$. This implies that $\eta = \theta_i\rho$ for some substitution ρ. We now distinguish the cases:

(i) $I \not\models_L B^-(r\eta)$, $I \not\models_L B^-(r_i\eta)$, $I \models_L B^+(r\eta) \cap DL^?_P$, and $I \models_L B^+(r_i\eta) \cap DL^?_P$: Here, $r\eta, r_i\eta \in sP^I_L$. By our unfolding rule, we get that $r'_i\eta \in (sP^I_L)'$. Since $r'_i \in \bar{P}'$, $\eta = \theta_i\rho$, and (i) hold, we conclude $r'_i\eta$ is in sP'^I_L.

(ii) for some $l \in B^-(r\eta) \cup B^-(r_i\eta)$, $I \models_L l$, or for some $a \in (B^+(r\eta) \cup B^+(r_i\eta)) \cap DL^?_P$, $I \models_L a$ hold: In this case, some of $r\eta$ and $r_i\eta$ is not in sP^I_L. Therefore, $r'_i\eta$ is not in $(sP^I_L)'$. Since $r'_i \in P'$, $\eta = \theta_i\rho$, and (ii) hold, we conclude $r'_i\eta$ is not in sP'^I_L either.

Thus, $sP'^I_L = (sP^I_L)'$, i.e., the reduct of the complete unfolded program P' and the complete unfolded reduct of P coincide. This implies I is a minimal model of $(L, (sP^I_L)')$ iff I is a minimal model of (L, sP'^I_L). Therefore, I is a strong answer set of (L, P'). $\square$

A.2.3 Proof Theorem 5.8

Proof. Algorithm 3 first copies P to P^l and applies Fact Pushing to P. Now suppose that we cannot do the Unfolding part of the algorithm, i.e., $C = \emptyset$ and only the Fact Pushing step takes part in the optimization process. $\mathit{merge}(P)$ eventually halts, since we cannot push any facts in P, therefore $P = P^l$. By part (D) of Theorem 5.6, Fact Pushing preserves the answer sets, hence $P \equiv_L \mathit{merge}(P, F)$.

Now assume that we unfold some rules in P, i.e., $C \neq \emptyset$. Some rules in P have a common atom $a \in C$ in the head and in the positive body, while a does not occur in the negative part of any rule in P. These a can be unfolded using the Unfolding rule (E). Algorithm 3 then proceeds by possibly unfolding all the rules $r \in R$ and $r' \in R'$ by means of $\mathit{unfold}(a, r, r')$, i.e., folding r into r' w.r.t. a. Since $\mathit{pred}(H(r)) \cap \mathcal{P} \neq \emptyset$, we always add r to P'. Thus, either $\mathit{unfold}(a, r, r') \cup \{r\}$ or $\{r, r'\}$ are contained in P', depending on the outcome of $\mathit{do\_unfold}$. Eventually, after all the unfolding had been carried out for a particular $a \in C$, we replace P by $P' \cup (P \setminus (R \cup R'))$, which amounts to replacing P by $(P \setminus \bar{P}) \cup \bar{P}'$ for all possible $\bar{P}$ and $\bar{P}'$, which are defined as in Theorem 5.6. Therefore, by part (E) of Theorem 5.6, one unfolding step for an $a \in C$ preserves the answer sets, hence after all other atoms of C had been unfolded, we still have the same answer sets as the program we started the unfolding procedure with. Ultimately, for this case, the unfolding procedure halts, since in each round of $\mathit{merge}(P, F)$'s main-loop, we check whether P equals P^l, the program P from which we started an optimization round, which indicates that no Unfolding or Fact Pushing could take place. □

A.2.4 Proof Theorem 5.9

We show now that $AS(P) = AS(\mathit{RuleOptimizer}(P))$.

Proof. Since $\mathit{RuleOptimizer}(P)$ takes each $r \in P$ and tries to optimize it, we have to check that each round of the main-loop preserves the answer sets.

For each r with $B^+(r) = \emptyset$, it is clear that no pushing can be performed, hence the answer sets remain the same.

For a rule r with dl-atoms in the positive body, i.e., with an arbitrary $b \in B^+(r)$, $I \models_L B^+(r)$ iff $I \models_L \mathit{BodyOptimizer}(b, B^+(r) \setminus \{b\}, \emptyset, \emptyset)$. Since the whole optimization procedure boils down to repetitive pushing of atoms via $\mathit{push}(o, b)$, we only have to check that o and b in contrast to $\mathit{push}(o, b)$ have the same answers over an arbitrary DL-KB L. We obtain that in a rule r with o, b in $B^+(r)$, we get a rule r' by replacing o, b in r with its optimized form $\mathit{push}(o, b)$. Thus, by Theorem 5.5, we immediately get that $AS(P) = AS((P \setminus \{r\}) \cup \{r'\})$.

The second loop in $\mathit{RuleOptimizer}(P)$ implements Variable Elimination by carefully taking each dl-atom in every rule of P into account, which has an atom $X = t$ in its CQ or in the rule body. Again, by Theorem 5.5, each replacement in the rules preserves the answer sets. □

Dilbert:	Wait, wait, with just one rat, we don't think any conclusions can be drawn.
Pointy-haired boss:	Oh, we'll draw conclusions, alright. You can be sure of that. Take this to the boys in the statistical distortion department. They'll fix the data for you.

—Dilbert, *The Fact*

B cq-Program Experiments: Setup and Results

The following tables B.1–B.6 show the outcome of the experiments described in Section 5.3. As reported there, we used the ontology testsuite from [Motik and Sattler, 2006] and the benchmark framework from [Guo et al., 2005]. In addition, [Motik and Sattler, 2006] provides some statistical information for the test ontologies used in their experiments. The corresponding test programs are shown in Table 5.2.

Our LUBM faculty experiment does not use the existing LUBM ontologies—the current version of RacerPro is not able to handle a LUBM ontology with two full universities and all their departments—, instead, fifteen test ontologies has been created out of the existing department ontologies using the LUBM data generator UBA 1.7 with applied Linux file path patch:[1]

```
java -cp UBA/classes edu.lehigh.swat.bench.uba.Generator -index 0 -seed 0
    -univ 1 -onto http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl.
```

Since the resulting ontologies import the Web-accessible LUBM class hierarchy, we carefully incorporated the contents of this TBox-only ontology at `http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl` into the generated OWL files to rule out the network latency in our experiment setup. Each resulting LUBM department ontology n include the LUBM Departments 1 up to n of LUBM University 1, for $1 \leq n \leq 15$. Following the statistical data presented in [Motik and Sattler, 2006], we will also report on statistical information for the generated LUBM ontologies. The statistics have been extracted using the OWL-Tools utility for the KAON2 DL-reasoner.[2] With this tool, the number of class axioms for ontology LUBM department 1–n have been counted using

```
owl dump lubm1-n.owl -ClassMember,
```

whereas object property axioms have been calculated using

```
owl dump lubm1-n.owl -ObjectPropertyMember.
```

Table B.7 summarized the number of ABox assertions for each of the LUBM ontologies.

All table entries are measured in seconds; missing entries ("—") indicate that the experiment timed out—the RacerPro process has been stuck in those situations—, or RacerPro used too much system resources.

[1] `http://projects.semwebcentral.org/projects/lubm/`

[2] `http://owltools.ontoware.org/`

wine_n	RacerPro	unoptimized program evaluation	RacerPro	optimized program evaluation
0	3.170	3.294	2.052	2.152
1	9.235	9.642	5.608	5.824
2	14.944	15.912	11.329	11.832
3	22.240	24.136	14.889	15.890
4	29.403	32.490	16.321	17.646
5	34.955	40.032	19.447	21.454
6	76.337	94.492	37.578	45.426
7	173.664	252.846	80.682	117.520
8	387.360	1006.394	170.196	427.008
9	—	—	378.121	1744.900
10	—	—	—	—

Table B.1: region experiment results

wine_n	RacerPro	unoptimized program evaluation	RacerPro	optimized program evaluation	RacerPro	optimized program evaluation
0	2.310	2.400	0.900	1.036	1.039	1.102
1	5.897	6.018	2.060	2.136	2.030	2.090
2	9.165	9.348	3.250	3.356	3.210	3.284
3	12.775	13.012	4.529	4.646	5.707	5.798
4	16.916	17.222	5.597	5.722	6.928	7.166
5	20.033	20.412	6.960	7.104	7.495	7.638
6	39.566	40.428	12.973	13.214	13.731	13.912
7	83.363	85.876	25.914	26.370	27.759	28.086
8	180.028	188.790	55.420	56.304	61.140	61.880
9	424.398	455.136	123.693	125.468	143.262	144.964
10	—	—	270.478	274.932	296.008	299.302

Table B.2: bigwinery experiment results

semintec_n	RacerPro	unoptimized program evaluation	RacerPro	optimized program evaluation
0	91.112	92.972	25.323	25.868
1	199.600	203.948	50.240	51.300
2	331.753	338.560	86.036	87.646
3	—	—	111.870	114.124
4	—	—	228.471	231.100

Table B.3: SEMINTEC experiment results

semintec_n	RacerPro	unoptimized program evaluation	RacerPro	optimized program evaluation
0	32.218	33.360	24.489	25.024
1	76.655	79.062	48.648	49.726
2	116.945	120.750	75.710	77.308
3	170.251	175.716	107.571	109.694
4	—	—	135.426	138.114

Table B.4: SEMINTEC costs experiment results

vicodi_n	RacerPro	unoptimized program evaluation	RacerPro	optimized program evaluation
0	10.826	13.256	9.182	10.220
1	26.209	31.234	18.802	20.186
2	—	—	34.755	36.684
3	—	—	47.396	49.730
4	—	—	64.998	67.868

Table B.5: VICODI experiment results

LUBM(1,0) Dept. 1–n	RacerPro	unoptimized program evaluation	RacerPro	optimized program evaluation
1	8.979	9.246	1.596	1.922
2	16.690	17.180	3.018	3.544
3	24.334	25.190	3.579	4.160
4	33.763	35.256	4.653	5.400
5	48.340	50.688	5.655	6.610
6	52.204	54.738	6.735	7.896
7	72.257	77.064	7.934	9.364
8	79.886	84.588	10.310	12.030
9	87.310	96.434	10.578	12.570
10	111.749	124.244	13.035	15.392
11	126.940	160.982	12.236	14.224
12	127.890	141.138	13.042	15.290
13	167.801	223.724	15.982	18.606
14	168.043	188.262	16.216	18.762
15	184.066	269.086	17.946	20.714

Table B.6: LUBM faculty experiment results

LUBM(1,0) Dept. 1–n	$C(a)$	$R(a,b)$
1	1623	4115
2	2883	7308
3	4070	10337
4	5297	13423
5	6511	16753
6	7752	20156
7	8782	22918
8	10050	26504
9	11339	30127
10	12363	32973
11	13575	36402
12	14813	39847
13	15920	42911
14	17185	46732
15	18128	49336

Table B.7: LUBM faculty ontology statistics

Bibliography

S. Abiteboul. Querying Semi-Structured Data. In F. N. Afrati and P. G. Kolaitis, editors, *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, volume 1186 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 1997. ISBN 3-540-62222-5. 1.2

S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 254–263, New York, NY, USA, 1998. ACM Press. URL `http://doi.acm.org/10.1145/275487.275516`. 2.8.1

S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison Wesley, Reading, US, 1995. 2.8, 4.5.2

K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. Scalable template-based query containment checking for web semantic caches. In *19th International Conference on Data Engineering (ICDE'03)*, pages 493–504, Los Alamitos, CA, USA, 2003. IEEE Computer Society. URL `http://www.cs.cmu.edu/~amiri/icde.pdf`. 4.5.2

G. Antoniou, C. V. Damásio, B. Grosof, I. Horrocks, M. Kifer, J. Maluszynski, and P. F. Patel-Schneider. Combining Rules and Ontologies: A survey. Technical Report IST506779/Linköping/I3-D3/D/PU/a1, Linköping University, February 2005. IST-2004-506779 REWERSE Deliverable I3-D3. 1.3.2, ii

K. Apt, H. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufman, Washington DC, 1988. 2.2.1

K. R. Apt. *Handbook of Theoretical Computer Science: Formal Models and Semantics*, volume B, chapter Logic Programming, pages 493–574. Elsevier, 1990. 1.1

F. Baader and C. Lutz. Description Logic. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *The Handbook of Modal Logic*, pages 757–820. Elsevier, 2006. URL `http://lat.inf.tu-dresden.de/research/papers/2006/BaLu-ML-Handbook-06.ps.gz`. 2.3

F. Baader and W. Snyder. *Handbook of Automated Reasoning*, chapter Unification Theory, pages 445–533. Elsevier, Sept. 2001. URL `http://lat.inf.tu-dresden.de/research/papers/2001/BaaderSnyderHandbook.ps.gz`. 2.1, 2.2, 2.3, 2.7.1

F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003. 1.3.1, 2.3, 3.2

F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ Envelope. In *Proc. of the Nineteenth Int. Joint Conf. on Art. Int. IJCAI-05*, Edinburgh, UK, 2005a. Morgan-Kaufmann. URL `http://lat.inf.tu-dresden.de/~clu/papers/archive/ijcai05.pdf`. 4.1.2

F. Baader, I. Horrocks, and U. Sattler. Description Logics as Ontology Languages for the Semantic Web. In D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg Siekmann on the Occasion of His 60th Birthday*, number 2605 in Lecture Notes in Artificial Intelligence, pages 228–248. Springer, 2005b. URL `http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/BaHS03.pdf`. 1.3.1

F. Baader, I. Horrocks, and U. Sattler. Description Logics. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007. URL `http://web.comlab.ox.ac.uk/oucl/work/ian.horrocks/Publications/download/2007/BaHS07a.pdf`. To appear. 2.3

S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference, 2004. URL `http://www.w3.org/TR/owl-ref/`. 2.4

D. Beckett and J. Grant. Mapping Semantic Web Data with RDBMSes. `http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/`, 2003. W3C Semantic Web Advanced Development for Europe (SWAD-Europe). 4.1.2

D. Berardi, D. Calvanese, and G. D. Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1):70–118, 2005. ISSN 0004-3702. URL `http://dx.doi.org/10.1016/j.artint.2005.05.003`. 2.3

T. Berners-Lee. Semantic Web Road Map, 1998. `http://www.w3.org/DesignIssues/Semantic.html`. 1.2

T. Berners-Lee and M. Fischetti. *Weaving the Web: The original design and ultimate destiny of the World Wide Web, by its inventor.* Harper Collins, 2000. 1.2

T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001. URL `http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&ref=sciam`. 1.2, 1.2

H. Boley, S. Tabet, and G. Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proceedings SWWS-2001*, 2001. 1.3.1

G. S. Boolos, J. P. Burgess, and R. C. Jeffrey. *Computability and Logic.* Cambridge University Press, fourth edition, 2003. 2

F. Bry and M. Marchiori. Ten Theses on Logic Languages for the Semantic Web. In *Proc. of the Third International Workshop Principle and Practice of Semantic Web Reasoning, PPSWR 2005*, volume 3703 of *LNCS*, pages 42–49. Springer, 2005. 1.3.2

D. Calvanese, G. D. Giacomo, and M. Lenzerini. On the Decidability of Query Containments under Constraints. In *Proceedings of PODS-98*, 1998. 2.8.1

D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable Description Logics for Ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005. URL `http://www.dis.uniroma1.it/~rosati/publications/Calvanese-etal-AAAI-05.pdf`. 2.8.1, 4.1.2

D. Calvanese, T. Eiter, and M. Ortiz. Answering Regular Path Queries in Expressive Description Logics: An Automata-Theoretic Approach. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2007)*, 2007a. URL `http://www.kr.tuwien.ac.at/staff/ortiz/pubs/calv-eite-orti-AAAI-2007.pdf`. 2.8.1

D. Calvanese, G. D. Giacomo, and M. Lenzerini. Conjunctive Query Containment and Answering under Description Logics Constraints. *ACM Transactions on Computational Logic (TOCL)*, 2007b. URL `http://www.acm.org/pubs/tocl/accepted/282calvanese.pdf`. To appear. 1.3.1, 2.8.1, 4.5.2, 5.1.3

A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90, May 1977. 2.8

P. P. Chen. The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976. 2.3

A. Colmerauer and P. Roussel. The birth of Prolog. In *HOPL-II: The second ACM SIGPLAN conference on History of programming languages*, pages 37–52, New York, NY, USA, 1993. ACM Press. ISBN 0-89791-570-4. URL `http://doi.acm.org/10.1145/154766.155362`. 1.1

B. Cuenca Grau, I. Horrocks, B. Parsia, P. Patel-Schneider, and U. Sattler. Next Steps for OWL. In *Proc. of the Second OWL Experiences and Directions Workshop*, volume 216 of *CEUR (http://ceur-ws.org/)*, 2006. URL `http://www.cs.man.ac.uk/~horrocks/Publications/download/2006/CHPP+06.pdf`. 1.2

J. de Bruijn, T. Eiter, A. Polleres, and H. Tompits. On Representational Issues About Combinations of Classical Theories with Nonmonotonic Rules. In *Knowledge Science, Engineering and Management*, volume 4092 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2006. URL `http://dx.doi.org/10.1007/11811220_1`. 1.3.2

J. de Bruijn, T. Eiter, A. Polleres, and H. Tompits. Embedding Non-Ground Logic Programs into Autoepistemic Logic for Knowledge-Base Combination. In *Proc. of the 20th Int. Joint Conf. on Art. Int. (IJCAI 2007)*, pages 304–309, Hyderabad, India, Jan. 2007a. AAAI. URL `http://www.ijcai.org/papers07/Papers/IJCAI07-047.pdf`. 1.3.2

J. de Bruijn, D. Pearce, A. Polleres, and A. Valverde. Quantified Equilibrium Logic and Hybrid Rules. In *First Int. Conference on Web Reasoning and Rule Systems (RR2007)*, volume 4524 of *LNCS*, Innsbruck, Austria, June 2007b. Springer. 1.3.2

J. Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In *Logic, Action and Information. Proceedings Konstanz Colloquium in Logic and Information* (*LogIn 1992*), pages 241–329. DeGruyter, 1995. 2.2

T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System dlv: Progress Report, Comparisons, and Benchmarks. In A. Cohn, L. Schubert, and S. Shapiro, editors, *Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 406–417, June 2–4 1998. 2.2

T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The Diagnosis Frontend of the `dlv` System. *The European Journal on Artificial Intelligence (AI Communications)*, 12(1–2):99–111, 1999. 1.1

T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving Using the DLV System. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, 2000. 2.2

T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. System Description: The DLV$^{\mathcal{K}}$ Planning System. In T. Eiter, W. Faber, and M. Truszczyński, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, number 2173 in LNCS, pages 429–433. Springer, 2001. System Description (abstract). 1.1

T. Eiter, M. Fink, H. Tompits, and S. Woltran. Simplifying Logic Programs under Uniform and Strong Equivalence. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Computer Science*, pages 87–99. Springer, 2004a. 2.7.2

T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the Semantic Web. In *Proceedings KR-2004*, pages 141–151, 2004b. Extended Report RR-1843-03-13, Institut für Informationssysteme, TU Wien, 2003. 1, 1.3.2, 2.5, 2.5, 3.1, 3.2, 3.3, 3.3.2, 7

T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Well-founded Semantics for Description Logic Programs in the Semantic Web. In G. Antoniou and H. Boley, editors, *Proceedings RuleML 2004 Workshop, ISWC Conference, Hiroshima, Japan, November 2004*, number 3323 in LNCS, pages 81–97. Springer, 2004c. 1, 2.5, 3.3, 4.3.2

T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Nonmonotonic Description Logic Programs: Implementation and Experiments. In F. Baader and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings*, volume 3452 of *Lecture Notes in Computer Science*, pages 511–517. Springer, 2005a. 4.5.2, 4.5.2

T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (*IJCAI 2005*). Morgan Kaufmann, 2005b. 1, 1.3.2, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, 2.14, 3.3.2, 4.2, 5, 5.3

T. Eiter, G. Ianni, A. Polleres, R. Schindlauer, and H. Tompits. Reasoning with Rules and Ontologies. In P. Barahona, F. Bry, E. Franconi, N. Henze, and U. Sattler, editors, *Reasoning Web, Summer School 2006*, number 4126 in LNCS, pages 93–127. Springer, 2006a. 1, 1.3.2, 1.3.2, ii, 3.1, 3.2, 3.3, 3.3.2, 7

T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Towards Efficient Evaluation of HEX Programs. In J. Dix and A. Hunter, editors, *Proceedings 11th International Workshop on Nonmonotonic Reasoning (NMR-2006), Answer Set Programming Track*, pages 40–46, May 2006b. Available as TR IfI-06-04, Institut für Informatik, TU Clausthal, Germany. ISSN 1860-8477. 1

T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. dlvhex: A Tool for Semantic-Web Reasoning under the Answer-Set Semantics. In A. Polleres, S. Decker, G. Gupta, and J. de Bruijn, editors, *Informal Proceedings Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS 2006), at FLOC/ICLP 2006, Seattle, WA, USA*, number 196 in CEUR Workshop Proceedings, pages 33–39, Aug. 2006c. URL `http://CEUR-WS.org/Vol-196/`. 4.4

T. Eiter, G. Ianni, T. Krennwallner, and R. Schindlauer. Exploiting Conjunctive Queries in Description Logic Programs. In D. Calvanese, E. Franconi, V. Haarslev, D. Lembo, B. Motik, A.-Y. Turhan, and S. Tessaris, editors, *Proceedings of the 20th International Workshop on Description Logics (DL2007)*, volume 250 of *CEUR Workshop Proceedings*, pages 259–266. CEUR-WS.org, June 2007a. URL `http://ceur-ws.org/Vol-250/paper_64.pdf`. 1, 3.1, 5

T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. Technical Report INFSYS RR-1843-07-04, Institut für Informationssysteme, TU Wien, Mar. 2007b. 1, 2.3, 2.5, 3.3.1, 6.1

W. Faber, N. Leone, and G. Pfeifer. Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. In *Proceedings JELIA-04*, pages 200–212, 2004. 2.6.2

D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuiness, and P. F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2): 38–45, 2001. 2.4

A. Ferrara, L. A. Ludovico, S. Montanelli, S. Castano, and G. Haus. A Semantic Web Ontology for Context-Based Classification and Retrieval of Music Resources. *ACM Transactions Multimedia Computing, Communications, and Applications (TOMCCAP*, 2(3):177–198, 2006. ISSN 1551-6857. URL `http://doi.acm.org/10.1145/1152149.1152151`. 6.2, 6.2

M. Garey and D. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979. 1.1

M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proceedings of the 5th International Conference and Symposium*, pages 1070–1080. MIT Press, 1988. 2.2

M. Gelfond and V. Lifschitz. Logic programs with classical negation. In D. Warren and P. Szeredi, editors, *Logic Programming: Proc. of the Seventh Int'l Conf.*, pages 579–597, Cambridge, MA, USA, 1990. MIT Press. 2.2

M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *NGC*, 9(3–4):365–386, 1991. 1.1, 2.1, 2.2, 2.6.2, 2.12

B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive Query Answering for the Description Logic $\mathcal{SHIQ}$. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 399–404. AAAI, Jan. 2007a. URL `http://www.ijcai.org/papers07/Papers/IJCAI07-062.pdf`. 2.8.1, 3.1, 3.3.2

B. Glimm, I. Horrocks, and U. Sattler. Conjunctive Query Entailment for $\mathcal{SHOQ}$. In *Proc. of the 20th International Workshop on Description Logics DL'07*, volume 250 of *CEUR Workshop Proceedings*, pages 65–80. CEUR-WS.org, June 2007b. 2.8.1, 3.3.2

B. C. Grau, D. Calvanese, G. D. Giacomo, I. Horrocks, C. Lutz, B. Motik, B. Parsia, and P. F. Patel-Schneider. OWL 1.1 Web Ontology Language Tractable Fragments. W3C Submission, World Wide Web Consortium (W3C), Dec. 2006. URL `http://www.w3.org/Submission/2006/SUBM-owl11-tractable-20061219/`. 2.8.1

B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logics. In *Proceedings WWW-2003*, 2003. 1.3.2

T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, Apr. 1993. URL `http://tomgruber.org/writing/ontolingua-kaj-1993.pdf`. 2.4

Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics*, 3(2–3):158–182, 2005. URL `http://dx.doi.org/10.1016/j.websem.2005.06.005`. 5.3, B

V. Haarslev and R. Möller. RACER System Description. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR-01)*, volume 2083 of *LNAI*, pages 701–705. Springer-Verlag, 2001. 5, 5.3

A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001. URL `http://dx.doi.org/10.1007/s007780100054`. 1.2

A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data Integration: The Teenage Years. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 9–16. ACM, 2006. ISBN 1-59593-385-9. URL `http://www.vldb.org/conf/2006/p9-halevy.pdf`. 1.2

J. Heflin and J. A. Hendler. Dynamic Ontologies on the Web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 443–449, Menlo Park, CA, 2000. AAAI Press / The MIT Press. ISBN 0-262-51112-6. 2.4

J. Hendler and D. L. McGuiness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6):67–73, 2000. 2.4

J. A. Hendler. The Semantic Web: KR's Worst Nightmare? In *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France.* Morgan Kaufmann, 2002. ISBN 1-55860-554-1. 1.2

I. Herman. Introduction to the Semantic Web. slides available at `http://www.w3.org/2007/Talks/0423-Stavanger-IH/`, April 2007. Speech at Semantic Days 2007. 1.2

H. Herre, B. Heller, P. Burek, R. Hoehndorf, F. Loebe, and H. Michalek. General Formal Ontology (GFO): A Foundational Ontology Integrating Objects and Processes. Part I: Basic Principles. Technical Report 8, Research Group Ontologies in Medicine (Onto-Med), University of Leipzig, 2006. URL `http://www.onto-med.de/en/publications/scientific-reports/om-report-no8.pdf`. 6.1

S. Heymans, D. V. Nieuwenborgh, and D. Vermeir. Preferential Reasoning on a Web of Trust. In *Proceedings of the Fourth International Semantic Web Conference, ISWC 2005, Galway, Ireland. LNCS 3729*, pages 368–382, 2005. 1.3.2

R. Hoehndorf, F. Loebe, J. Kelso, and H. Herre. Upper ontologies are not enough: The Role of nonmonotonic reasoning in the integration of biomedical ontologies. 2007. Manuscript submitted for puplication. 6.1, 6.1, 6.1

I. Horrocks. DAML+OIL: A Reason-able Web Ontology Language. In *Proceedings EDBT-2002*, volume 2287 of *LNCS*, pages 2–13. Springer, 2002a. 2.4

I. Horrocks. DAML+OIL: A Description Logic for the Semantic Web. *IEEE Bulletin of the Technical Committee on Data Engineering*, 25(1):4–9, 2002b. 2.4

I. Horrocks. Description Logics in Ontology Applications. In B. Beckert, editor, *Proc. of the 9th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2005)*, number 3702 in Lecture Notes in Artificial Intelligence, pages 2–13. Springer, 2005. ISBN 3-540-28931-3. URL `http://www.cs.man.ac.uk/~horrocks/Publications/download/2005/Horr05b.pdf`. 2.4

I. Horrocks and P. F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. In *Proc. ISWC-2003*, volume 2870 of *LNCS*, pages 17–29. Springer, 2003. 2.3

I. Horrocks and U. Sattler. Ontology Reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic. In *Proceedings IJCAI-01*, pages 199–204, 2001. 2.3

I. Horrocks and U. Sattler. A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005. URL `http://www.cs.man.ac.uk/~horrocks/Publications/download/2005/HoSa05a.pdf`. 2.3

I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In *Proceedings LPAR-1999*, volume 1705 of *LNCS*, pages 161–180. Springer, 1999. 2.3.1

I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, 2003. 1.2, 2.3, 2.4, 2.4

I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, May 2004. W3C Member Submission. `http://www.w3.org/Submission/SWRL/`. 1.3.1, 1.3.2, 6.2

I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler. Semantic Web Architecture: Stack or Two Towers? In *Proceedings of the Third International Workshop Principle and Practice of Semantic Web Reasoning, PPSWR 2005*, volume 3703 of *LNCS*, pages 37–41. Springer, 2005. 1.2

U. Hufstadt, B. Motik, and U. Sattler. Reasoning for Description Logics around $\mathcal{SHIQ}$ in a Resolution Framework. Technical Report 3-8-04/04, Forschungszentrum Informatik (FZI), Karlsruhe, 76131 Karlsruhe, Germany, July 8, 2004. 1.3.2

U. Hustadt, B. Motik, and U. Sattler. Data Complexity of Reasoning in Very Expressive Description Logics. In *Proceedings of Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 466–471, 2005. URL `http://www.cs.man.ac.uk/~bmotik/publications/papers/hms05data.pdf`. 2.8.1

M. Knorr, J. J. Alferes, and P. Hitzler. A Well-founded Semantics for Hybrid MKNF Knowledge Bases. In *Proc. of 20th International Workshop on Description Logics DL'07*, volume 250 of *CEUR Workshop Proc.*, pages 347–354. CEUR-WS.org, 2007. URL `http://CEUR-WS.org/Vol-250/paper_54.pdf`. 2.5

R. Kowalski. Algorithm = Logic + Control. *Commun. ACM*, 22(7):424–436, 1979. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/359131.359136`. 1.1

R. A. Kowalski. Predicate Logic as a Programming Language. In *Proceedings IFIP'74*, pages 569–574. North-Holland, 1974. 1.1

A. Krisnadhi and C. Lutz. Data Complexity in the $\mathcal{EL}$ family of DLs. In *Proceedings of the 20th International Workshop on Description Logics (DL-2007)*, volume 250 of *CEUR-WS Online Proceedings*, pages 88–99. CEUR-WS.org, June 2007. URL `http://CEUR-WS.org/Vol-250/paper_15.pdf`. 2.8.1

M. Krötzsch and S. Rudolph. Conjunctive Queries for $\mathcal{EL}$ with Composition of Roles. In *Proceedings of the 20th International Workshop on Description Logics (DL-2007)*, volume 250 of *CEUR-WS Online Proceedings*, pages 355–362. CEUR-WS.org, June 2007. URL `http://CEUR-WS.org/Vol-250/paper_58.pdf`. 2.8.1

M. Krötzsch, S. Rudolph, and P. Hitzler. Conjunctive Queries for a Tractable Fragment of OWL 1.1. In *Proc. ISWC-2007*, 2007. URL `http://korrekt.org/papers/KroetzschRudolphHitzler_ELquerying_ISWC2007.pdf`. To appear. 2.8.1

N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006. ISSN 1529-3785. URL `http://doi.acm.org/10.1145/1149114.1149117`. 2.2

A. Y. Levy and M.-C. Rousset. Combining Horn Rules and Description Logics in CARIN. *Artif. Intell.*, 104(1-2):165–209, 1998. 1.3.2, 1.3.2, 2.8.1

V. Lifschitz and T. Woo. Answer Sets in General Nonmonotonic Reasoning (Preliminary Report). In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning* (*KR 1992*), pages 603–614. Morgan Kaufmann, 1992. 2.2

T. Lukasiewicz. Stratified Probabilistic Description Logic Programs. In *Proc. ISWC 2005 Workshop on Uncertainty Reasoning for the Semantic Web*, pages 87–97, 2005. 1.3.2

T. Lukasiewicz. A Novel Combination of Answer Set Programming with Description Logics for the Semantic Web. In *Proceedings European Conference on Semantic Web (ESWC 2007)*, LNCS. Springer, June 2007. 1.3.2, 4.3.2, 4.2

F. Manola and E. Miller. RDF Primer. W3c recommendation, World Wide Web Consortium (W3C), Cambridge, Massachusetts, United States, 2004. URL `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`. 1.2

D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, World Wide Web Consortium (W3C), Feb. 2004. URL `http://www.w3.org/TR/2004/REC-owl-features-20040210/`. 1.2

B. Motik and R. Rosati. A Faithful Integration of Description Logics with Logic Programming. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 477–482, Hyderabad, India, January 6–12 2007. AAAI. URL `http://www.ijcai.org/papers07/Papers/IJCAI07-075.pdf`. 1.3.2, 3.3.2

B. Motik and U. Sattler. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In *Proc. LPAR 2006*, volume 4246 of *LNCS*, pages 227–241. Springer, November 13–17 2006. 5.3, 5.3, B

B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In *Proceedings of the 3rd International Semantic Web Conference (ISWC 2004)*, pages 549–563, 2004. 1.3.1, 1.3.2

B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, 2005. 1.3.2, 3.1

B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and Logic Programming Live Together Happily Ever After? In *Proceedings of the 2006 International Semantic Web Conference (ISWC 2006)*, volume 4273 of *Lecture Notes in Computer Science*, pages 501–514. Springer, 2006. URL `http://www.cs.man.ac.uk/~horrocks/Publications/download/2006/MHRS06.pdf`. 1, 1.3.1, 1.3.2, 3.3.2

B. Motik, I. Horrocks, and U. Sattler. Bridging the Gap Between OWL and Relational Databases. In *Proceedings of the Sixteenth International World Wide Web Conference (WWW 2007)*, 2007a. URL `http://www.cs.man.ac.uk/~horrocks/Publications/download/2007/MoHS07a.pdf`. 1.3.1

B. Motik, I. Horrocks, and U. Sattler. Adding Integrity Constraints to OWL. In C. Golbreich, A. Kalyanpur, and B. Parsia, editors, *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007b. URL `http://ceur-ws.org/Vol-258/paper11.pdf`. 1.3.1

I. Niemelä. Logic Programming with Stable Model Semantics as Constraint Programming Paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):241–273, 1999. 2.2

A. Nonnengart and C. Weidenbach. Computing Small Clause Normal Forms. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science, 2001. URL `http://www.mpi-inf.mpg.de/~weidenb/publications/handbook99small.ps.gz`. A.2

M. Ortiz de la Fuente, D. Calvanese, and T. Eiter. Characterizing Data Complexity for Conjunctive Query Answering in Expressive Description Logics. In *Proc. AAAI '06*. AAAI Press, 2006a. 2.8.1, 3.1, 3.3.2

M. Ortiz de la Fuente, D. Calvanese, and T. Eiter. Data Complexity of Answering Unions of Conjunctive Queries in $\mathcal{SHIQ}$. In *Proc. DL2006*, number 189 in CEUR Workshop Proceedings, pages 62–73, 2006b. 3.1, 3.3.2

J. Z. Pan, E. Franconi, S. Tessaris, G. Stamou, V. Tzouvaras, L. Serafini, I. R. Horrocks, and B. Glimm. Specification of Coordination of Rule and Ontology Languages. Project Deliverable D2.5.1, KnowledgeWeb NoE, June 2004. 1.3.2, ii

P. Patel-Schneider. A Revised Architecture for Semantic Web Reasoning. In *Proceedings of the Third International Workshop Principle and Practice of Semantic Web Reasoning, PPSWR 2005*, volume 3703 of *LNCS*, pages 32–36. Springer, 2005. 1.2

P. F. Patel-Schneider and I. Horrocks. Position Paper: A Comparison of Two Modelling Paradigms in the Semantic Web. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 3–12, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-323-9. doi: http://doi.acm.org/10.1145/1135777.1135784. 1.3.2

P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3c recommendation, World Wide Web Consortium (W3C), Cambridge, Massachusetts, United States, 2004. URL `http://www.w3.org/TR/2004/REC-owl-semantics-20040210/`. 1.2, 2.4

RacerPro Reference Manual Version 1.9. Racer Systems GmbH & Co. KG, December 2005a. URL `http://www.racer-systems.com/products/racerpro/reference-manual-1-9.pdf`. 4.3, 4.3.1, 4.3.3

RacerPro Users Guide Version 1.9. Racer Systems GmbH & Co. KG, December 2005b. URL `http://www.racer-systems.com/products/racerpro/users-guide-1-9.pdf`. 4.3, 4.3.1, 4.3.2, 4.3.3

R. Reiter. On Closed World Data Bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, New York, 1978a. 1.3.2

R. Reiter. A Logic for Default Reasoning. *Artificial Intelligence*, 13:81–132, 1980. 2.2, 6.1

R. Reiter. On Reasoning by Default. In *Proceedings of the 1978 workshop on Theoretical issues in natural language processing*, pages 210–218, Morristown, NJ, USA, 1978b. Association for Computational Linguistics. URL `http://dx.doi.org/10.3115/980262.980297`. 1.3.1

R. Reiter. What Should A Database Know? In *PODS '88: Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 302–304, New York, NY, USA, 1988. ACM Press. URL `http://doi.acm.org/10.1145/308386.308462`. 1.3.1

R. Rosati. Towards expressive KR systems integrating datalog and description logics: preliminary report. In *Proceedings of the 1999 International Workshop on Description Logics (DL-1999)*, pages 160–164, 1999. 1.3.2

R. Rosati. On Conjunctive Query Answering in $\mathcal{EL}$. In *Proceedings of the 20th International Workshop on Description Logics (DL-2007)*, volume 250 of *CEUR-WS Online Proceedings*, pages 451–458. CEUR-WS.org, June 2007a. URL `http://CEUR-WS.org/Vol-250/paper_83.pdf`. 2.8.1

R. Rosati. On the Decidability and Complexity of Integrating Ontologies and Rules. *Journal of Web Semantics*, 3(1):61–73, 2005a. 1.3.2

R. Rosati. $\mathcal{DL}$+*log*: Tight Integration of Description Logics and Disjunctive Datalog. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 68–78. AAAI Press, 2006a. 1.3.2, 3.3.2

R. Rosati. Integrating Ontologies and Rules: Semantic and Computational Issues. In *Reasoning Web, Summer School 2006*, number 4126 in LNCS, pages 128–151. Springer, 2006b. 1, 1.3.2, 1.3.2, ii, 3.1, 3.3.2

R. Rosati. The Limits of Querying Ontologies. In *Proceedings of the Eleventh International Conference on Database Theory (ICDT 2007)*, volume 4353 of *LNCS*, pages 164–178. Springer, 2007b. URL `http://dx.doi.org/10.1007/11965893_12`. 2.8.1, 3.1, 5.1.3

R. Rosati. Semantic and Computational Advantages of the Safe Integration of Ontologies and Rules. In *Proceedings of the Third International Workshop on Principles and Practice* *of Semantic Web Reasoning (PPSWR 2005)*, volume 3703 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2005b. 1.3.2, 1.3.2

K. A. Ross. Modular Stratification and Magic Sets for Datalog Programs with Negation. *J. ACM*, 41(6):1216–1266, 1994. 2.6.2, 2.12

J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison Wesley Professional, 2nd edition, 2004. 4.1.2

J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language User Guide.* Addison Wesley Professional, 2nd edition, 2005. 4.1.2

C. Sakama and H. Seki. Partial Deduction of Disjunctive Logic Programs: A Declarative Approach. In L. Fribourg and F. Turini, editors, *Logic Programming Synthesis and Transformation, Meta-Programming in Logic: Fourth International Workshops, LOBSTR'94 and META'94, Pisa, Italy*, pages 170–182. Springer, Berlin, 1994. URL `http://citeseer.ist.psu.edu/sakama94partial.html`. 2.7.2

C. Sakama and H. Seki. Partial Deduction in Disjunctive Logic Programming. *Journal of Logic Programming*, 32(3):229–245, 1997. URL `http://dx.doi.org/10.1016/S0743-1066(96)00120-3`. 2.7.2, 2.5, 2.19, 2.20, 2.21, 5.1.5

R. Schindlauer. *Answer-Set Programming for the Semantic Web.* PhD thesis, Technische Universität Wien, 12 2006. 1, 2.7, 2.6.2, 4.1

P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138:181–234, June 2002. 2.2

E. Sirin and B. Parsia. Optimization for Answering Conjunctive ABox Queries: First Results. In B. Parsia, U. Sattler, and D. Toman, editors, *Proc. of the 2006 International Workshop on Description Logics (DL2006)*, volume 189 of *CEUR Workshop Proc.* CEUR-WS.org, 2006. URL `http://ceur-ws.org/Vol-189/submission_32.pdf`. 5.2.3

E. Sirin and B. Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In C. Golbreich, A. Kalyanpur, and B. Parsia, editors, *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, volume 258 of *CEUR Workshop Proceedings.* CEUR-WS.org, 2007. URL `http://ceur-ws.org/Vol-258/paper14.pdf`. 4.1.2

B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, 3rd edition, 1997. 4

R. Tennant. The 5th wave comic strip. Universal Press Syndicate, 9/19 2004. (document)

G. Tzanetakis and P. Cook. Musical Genre Classification of Audio Signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002. URL `http://www.cs.uvic.ca/~gtzan/work/pubs/tsap02gtzan.pdf`. 6.2

D. van Dalen. *Logic and Structure.* Universitext. Springer, fourth edition, 2004. 2

R. van der Meyden. *Logics for Databases and Information Systems*, chapter Logical Approaches to Incomplete Information: A Survey, pages 309–358. Kluwer, 1998. 1.3.1, 2.8

A. Van Gelder, K. A. Ross, and J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991. 2.5

M. Y. Vardi. Why is Modal Logic So Robustly Decidable? In N. Immerman and P. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *Discrete Mathematics and Theoretical Computer Science*, pages 149–184. American Mathematical Society, 1997. 1.3.1

K. Wang, G. Antoniou, R. W. Topor, and A. Sattar. Merging and Aligning Ontologies in dl-Programs. In A. Adi, S. Stoutenburg, and S. Tabet, editors, *Proceedings of the First International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2005), Galway, Ireland*, volume 3791 of *Lecture Notes in Computer Science*, pages 160–171. Springer, 2005. 4.4

M. Wessel and R. Möller. A Flexible DL-based Architecture for Deductive Information Systems. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *Proc. IJCAR-06 Workshop on Empirically Sucessful Computerized Reasoning (ESCoR)*, pages 92–111, 2006. 4.3.1