FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Inline Evaluation of Hybrid Knowledge Bases

PhD THESIS

submitted in partial fulfillment of the requirements of

Doctor of Technical Sciences

within the

Vienna PhD School of Informatics

by

Guohui Xiao
Registration Number 0928566

to the Faculty of Informatics
at the Vienna University of Technology

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

Wien, 02.12.2013

(Signature of Author)

(Signature of Advisor)

Declaration of Authorship

Guohui Xiao
Piazza Domenicani 3, I-39100 Bolzano-Bozen BZ, Italy

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

Bozen, 2 December 2013

(Place, Date) (Signature of Author)

To my parents Fengchong Xiao and Shuhui Yan.

献给我的父亲肖凤冲和母亲闫淑慧

Acknowledgements

First of all, I want to thank my advisor Prof. Thomas Eiter. It was a great and exciting experience being his student and working with him. As a top researcher, he had a very busy schedule, but he was always devoting his time to his students for discussions, comments, meetings and encouragements. Even when I was abroad, he managed two face-to-face meetings with me during his vacations to finalize this thesis.

I am grateful to my reviewers Prof. Diego Calvanese and Prof. Sebastian Ruldoph for the helpful discussions and constructive suggestions.

I want to thank the Vienna PhD School of Informatics, which provided me the opportunity for this PhD study. Prof. Hannes Werthner and Prof. Hans Tompits have been doing a great job in organizing and leading the PhD school. The secretaries Clarissa Schmid and Mamen Calatrava Moreno supported me in many aspects during the PhD study from the visa application to the final defense.

It was a great experience working in the Knowledge Based System group in the Vienna University of Technology. Stijn Heymans guided me to start the research and taught me how to do the research professionally. I had many happy and productive collaborations with Magdalena Ortiz, Mantas Simkus, Thomas Krennwallner, Patrik Schneider, and Cristina Feier, resulting in several joint publications. Peter Schüller, Thomas Krennwallner and Christoph Redl developed the great DLVHEX solver and always provided me technical supports. Our secretary Eva Nedoma and technician Matthias Schlögel were always helpful when I needed them. Thanks to all the people in the KBS and DBAI groups for the last four years.

My gratitude also goes to all who inspired me during my PhD. The lectures from Georg Gottlob, Thomas Eiter, Reinhard Pichler and Axel Polleres gave me deep understandings of the database theory, the complexity theory, and the Semantic Web technologies, which are the foundations of this thesis. In the DLV team, Francesco Ricca, Wolfgang Faber and Nicola Leone developed the DLV system and always answered all my questions related to the system. Worarat Krathu provided the use cases and benchmarks from the business domain used in the thesis. Furthermore, I thank my master supervisor Zuoquan Lin who brought me into the field of AI and KR and encouraged me doing the PhD. I thank Guilin Qi, Yue Ma, Xiaowang Zhang and Kedian Mu for their encouragements and the happy collaborations on a variety of research topics.

I also want to thank my Chinese friends in Vienna and Munich. They let me feel that I am at home. With them, the life was much more interesting.

Finally, I want to thank my family and in particular my parents for their loves and understandings. My special thanks goes to, of course, my girlfriend Linfang Ding. Without her support and love, this thesis would never exist.

This thesis was partially funded by the EU Project OntoRule (IST-2009-231875), Austrian Science Fund (FWF) Project Reasoning in Hybrid Knowledge Bases (P20840), and the Vienna PhD School of Informatics.

Abstract

The deployment of knowledge representation formalisms to the Web has created the need for hybrid formalisms that combine heterogeneous knowledge bases (KBs). There are many formalisms proposed by the knowledge representation community for modeling knowledge bases, among which two families are of great importance, in particular in the context of semantic Web. One is the family of Description Logics (DLs) based ontologies, and the other is rule-based logic programming.

The challenge of combining ontologies and rules has been drawing a lot of attentions in recent years. Among several proposed approaches, loose coupling of rules and ontologies aims at combining respective knowledge bases by means of a clean interfacing semantics, in which roughly speaking inferences are mutually exchanged such that the one KB takes the imported information into account, and exports in turn conclusions to the other KB. This approach is fostered by non-monotonic description logic (dl-) programs, where this exchange is handled by a generalization of the answer set semantics of non-monotonic logic programs.

Because of the loose coupling nature of dl-programs, one can build engines for dl-programs on top of legacy reasoners. For instance, the DLVHEX system with dl-plugin, which is a state-of-the-art system for dl-programs, is built on top of the ASP reasoner DLV and the DL reasoner RacerPro. Although this architecture is very elegant, the performance of this implementation is suboptimal. We observe that the overhead of calling external reasoners in the classical approach can be the bottleneck of the performance.

The aim of this thesis is to improve the reasoning efficiency over hybrid KBs. We propose a new strategy, called inline evaluation, which compiles the whole hybrid KB into a new KB using only one single formalism. Hence we can use a single reasoner to do the reasoning tasks, and improve the efficiency of hybrid reasoning. In case of dl-programs, we design an abstract framework rewriting dl-programs to Datalog programs with negation, by compiling all components carefully and combining them together into a single program. The reduction is sound and complete when the DL ontologies are "Datalog-rewritable". We show that many DLs are Datalog-rewritable by introducing concrete rewriting algorithms. Furthermore, we show that inline evaluation can be used in hybrid KBs of other formalisms.

To confirm the hypothesis that the inline evaluation is superior to the classical approach of "ASP + external DL reasoner", we implement the inline evaluation method in the novel DReW system for dl-programs. We conduct an extensive evaluation on several benchmark suites and show that DReW outperforms the classical approach in general, especially for dl-programs of complex structure or with large instances.

Contents

CHAPTER 1

Introduction

1.1 Combining Rules and Ontologies

In the last decade, the growing importance of the (Semantic) Web and its envisioned future development has triggered a lot of research on accessing and processing data based on semantic approaches. Knowledge representation (KR) provides solid theoretical foundations for these approaches and new research topics for KR are raised when applying these semantics techniques.

A KR formalism is normally formally defined by a logic with well-defined syntax and clear semantics. For each formalism, the research can be roughly divided into theoretical and practical directions. The former concerns the theoretical properties of the logics, e.g., expressivity and complexity; the latter is about the realization of the formalism by developing and optimizing the reasoning algorithms, and implementing them in software programs of so-called reasoners.

There are many formalisms proposed by the KR community for modeling knowledge bases (KBs), among which the two families are of importance, in particular in the context of Semantic Web. One family is Description Logics (DLs), and the other is logic programming. They are studied intensively and implemented in many practical systems. Moreover, they are two important building blocks in the architecture of the Semantic Web layered stack and are standardized by the W3C.

Description Logics (DLs) are mostly fragments of first-order logic with a clear-cut semantics, convenient syntax and decidable reasoning, performed by quite efficient algorithms [Baa+07]. They range from tractable DLs (e.g., RL, EL, and DL-Lite), over expressive DLs (e.g., Horn-$\mathcal{SHIQ}$) to very expressive ones (e.g., $\mathcal{SHIQ}$, $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$). DLs are the theoretical foundation of the W3C Web Ontology Language (OWL). Because of the close relation of DL and OWL, KB formulated in DLs are often called ontologies. On the practical side, many efficient OWL reasoners (e.g., Pellet, RacerPro, HermiT) are implemented for DLs.

Logic programming is a family of rule-based languages, among that are Datalog, Datalog with negations (Datalog$^{\neg}$), and Datalog with disjunctions (Datalog$^{\vee}$). Unlike DL families with a standard first-order semantics, several semantics are proposed for logic programmings, among which two most important ones are the stable model semantics [GL88] and the well-founded

semantics [GRS91]. The stable models of a program are also called answer sets [GL91], and the logic programming under stable models semantics is also called answer set programming (ASP), which is implemented in many ASP solvers, such as DLV and Clasp.

The distributed nature of the Web poses a challenge for integrating heterogeneous data sources. In KR terms, this means a need for hybrid KBs combining KBs formulated in different logics. In the context of Semantic Web, we focus on combining description logic based ontologies and logic programs. Such combination is non-trivial because simply putting them together easily leads to high computational complexities or even undecidability [LR98]. Normally, some forms of syntax or semantic restrictions on the interaction between the ontology and rule components have to be applied for retaining a reasonable complexity, but this comes at the cost of sacrificing expressivity.

The challenge of combining ontologies and rules has been drawing a lot of attentions in recent years. The approaches of hybrid knowledge bases fall into three categories following the representational paradigms of the respective approaches [de +09]:

(1) The *loose coupling approaches* define an interface between the two formalisms based on *the exchange of the entailed knowledge*.

(2) The *tight coupling approaches* define an interface based on *common models*.

(3) The *embedding approaches* define an interface based on embeddings of both the ontology and the rules in *a single unifying non-monotonic formalism*.

Among several approaches, loose coupling of rules and ontologies aims at combining respective knowledge bases by means of a clean interfacing semantics, in which roughly speaking inferences are mutually exchanged such that the one KB takes the imported information into account, and exports in turn conclusions to the other KB. This approach is fostered by nonmonotonic description logic (dl-) programs [Eit+08a], where this exchange is handled by a generalization of the answer set semantics of nonmonotonic logic programs [GL91]. Follow up work has adapted this approach to other formalisms (e.g., [Wan+04; Hey+10; Eit+05]) and considered alternative semantics (e.g. [Luk10; Wan+10; Eit+11]).

The loose coupling approach is attractive in several regards. First, legacy KBs, powered by different reasoning engines, can be combined. Second, thanks to the interfacing and loose semantics connection, it is fairly easy to incorporate further knowledge formats besides rules and OWL (description logic) ontologies, e.g. RDF KB. HEX-programs [Eit+05] are a respective generalization of dl-programs, which in fact allow for incorporating arbitrary software. Third, view based data access of loose coupling is in support of privacy, as the internal structure of a KB remains hidden.

On the other hand, the impedance mismatch of different formalisms and reasoning engines comes at a price. A simple realization of the loose coupling considers the interface calls as an API which makes computation expensive, in particular if rules lead to choices via the underlying semantics. The black box view of other KBs hinders optimization and is a major obstacle for scalability.

We try to improve the efficiency of reasoning over hybrid KBs by borrowing the concepts of inline expansion in the programming language community. Inline expansion is an optimization

that replaces the function call with the actual body of the called function, which removes the costs of the function calls and return instructions. More importantly, since the body is directly accessible, further optimization is possible. The situation of the dl-programs is conceptually similar. The overhead of calling external reasoners can be a performance bottleneck.

1.2 Contributions

The goal of this thesis is to provide efficient novel reasoning methods for hybrid knowledge bases, in particular for dl-program. We propose the "inline evaluation" framework for hybrid KBs, which converts the evaluation problem of hybrid KBs into an (equivalent) KB in one formalism for evaluation and thus eliminates the overhead of calling external reasoners. The hypothesis is that the inline evaluation outperforms the classical approach of simply combining legacy reasoners to access the underlying data sources.

The contributions of this thesis are on both the theoretical and the practical side. On the theoretical side, we develop an "inline evaluation" framework for dl-programs by reducing them to Datalog$^\neg$ programs and show that this method can be applied or extended to some other hybrid KBs.

- We abstractly define the framework of inline evaluation for dl-programs by reducing them to Datalog$^\neg$ programs. Essentially, we use the Datalog$^\neg$ program to simulate the different components of the dl-programs, including the ontology component, the rule component and even their interactions. The reduction requires that DL ontologies are "Datalog-rewritable", that is, the ontology can be rewritten to an equivalent Datalog program. We propose a concrete Datalog-rewritable DL $\mathcal{LDL}^+$ and find that it is strongly related to OWL 2 RL. We also show that the DLs $\mathcal{SROEL}(\sqcap, \times)$ (the logic underpinning OWL 2 EL), and Horn-$\mathcal{SHIQ}$ are Datalog-rewritable by introducing concrete rewriting algorithms.

- We apply or extend inline evaluation to several other hybrid KBs.

  - Terminological default logic KBs are a combination of default logics and DL ontologies [BH95]. We adopt the encoding to dl-programs and showed that they can be inline evaluated when the ontology part is Datalog rewritable.
  - Similarly, dl-safe conjunctive queries (CQs) [MSS05] can be reduced to dl-programs easily and then inline evaluation could be applied.
  - We extended the query rewriting algorithm to general conjunctive queries over expressive DL Horn-$\mathcal{SHIQ}$, and more general weakly dl-safe KBs.
  - cq-programs are an extension of dl-programs using CQs to access ontologies [Eit+08b]. We extend the inline evaluation to cq-programs using the query rewriting techniques.

On the practical side, we implemented a new system for dl-programs and conducted extensive evaluations.

- We implement the inline evaluation of dl-programs in the DReW system, which also supports dl-safe conjunctive queries and terminological default theories.

- We create several novel benchmark suites for dl-programs and evaluate the DReW system and DLVHEX system on these benchmark suites.

Recall that the hypothesis was that inline evaluation can be much more efficient than the classical approach. The results show that inline evaluation is a general framework for reasoning over several hybrid KBs. The implementation and evaluation confirm that the DReW system outperforms DLVHEX in general, especially for dl-programs of complex structure or dl-programs with large instances.

We also note that inline evaluation is not always feasible. For instance, when ontologies can only be accessed through a query interface, the current approach cannot transform the ontologies to Datalog programs.

1.3 Evolution of this Work

In this section, we describe the evolution of this work and spot relevant publications. Most of the work of this thesis has been carried in the context of EU Project OntoRule and FWF Project Reasoning in Hybrid KBs.

2009 – 2010 This work started in October 2009. The goal of developing new efficient reasoning methods for hybrid KBs and implementing systems was very clear from the beginning. The initial work was to identify a tractable fragment of dl-programs under well-founded semantics and to show that reasoning can be done by a reduction to Datalog$^{\neg}$. The first result was published at ECAI 2010 [HEX10]. After that, we implemented the the first version of DReW system in two months and presented it as a workshop paper in BuRO 2010 [XHE10].

2011 – 2012 Based on the established results, the framework of inline evaluation became mature and was written down as the PhD proposal and later published in RR 2011 [XE11]. We continued the work on dl-programs and we found that the family of EL ontologies can also be faithfully handled in our framework and it can be used for terminological default logics. These results were parts of an invited paper in FoIKS 2012 [Eit+12a]. The implementation of DReW was further refined and the new version was described in a paper at SWWS 2012 [XEH12].

In parallel to the research on dl-programs, we also investigated conjunctive queries over the description logic Horn-$\mathcal{SHIQ}$. We developed a query rewriting algorithm for CQ over Horn-$\mathcal{SHIQ}$ by a reduction to Datalog and implemented it in the clipper system. The results were published at AAAI 2012 [Eit+12b] and DL 2012 [Eit+12c].

2013 In 2013, our main goal has been to converge the work and finish the thesis writing. To make the results complete, we spent quite some efforts on building several novel benchmarks and did extensive evaluations.

1.4 Thesis Organization

The rest of this thesis is structured as follows:

- In Chapter 2, we introduce all the preliminary knowledge for the thesis. We start from first-order logic and computational complexity theory. Then we recall two important knowledge representation languages, namely description logics (ontologies) and logic programming (rules).

- In Chapter 3, we give a brief survey on the state of the art of the approaches of combination of ontologies and rules. In particular, we focus on dl-programs, which is a loose coupling approach of combining ontologies and rules.

- In Chapter 4, we present the main theoretical contribution of this thesis. We first present the general framework of inline evaluation of dl-programs, which reduces the reasoning over dl-programs to that over Datalog$^{\neg}$. Then we show how to apply this framework to dl-programs over different Datalog-rewritable description logics: $\mathcal{LDL}^{+}$ (closely related to OWL 2 RL), OWL 2 EL, and up to Horn-$\mathcal{SHIQ}$.

- In Chapter 5, we apply the idea of inline evaluation to other formalisms of hybrid knowledge bases: terminological default logics, dl-safe conjunctive queries, general conjunctive queries and weakly dl-safe rules over Horn-$\mathcal{SHIQ}$, and cq-programs.

- In Chapter 6, we present the DReW system, a dl-programs reasoner based on the inline evaluation strategy. When reasoning, it first rewrites the whole dl-program into a Datalog$^{\neg}$ program, and then calls a Datalog reasoner (currently DLV) for the underlying reasoning.

- In Chapter 7, we present the evaluation results of the inline evaluation approach. The evaluations are performed on several novel benchmark suites. The results show that the inline evaluation approach outperforms the classical one in most of the tests.

- Finally in Chapter 8, we summarize this thesis and discuss issues and directions for future work.

CHAPTER 2

Preliminaries

In this chapter, we introduce some necessary knowledge for the thesis. We first recall first-order logic and computational complexity theory, and then we investigate two families of widely used knowledge representation languages: (description logic based) ontologies and (Datalog with negation) rules. Finally, we compare these two families of logics.

2.1 First-order Logic

First-order logic is the fundamental of all the logics used later in this thesis. There are many introductions and textbooks for first-order logic; readers may refer to [Ros53; Bry+07; Rau09].

Definition 2.1 (First-order vocabulary and variable)**.** *A first order vocabulary is a pair* $(\mathsf{N_P}, \mathsf{N_F})$ *of countably disjoint sets. The elements of* $\mathsf{N_P}$ *are* predicate symbols*, usually denoted by* p, q, r*; the elements of* $\mathsf{N_F}$ *are called* function symbols*, denoted by* f, g*. Each predicate and function is associated with a natural number, called* arity*. We assume* $\mathsf{N_P}$ *contains two special 0-ary predicates* $\top$ *and* $\bot$*, called top and bottom.*

We use a set $\mathsf{N_V}$ *of variables; the elements are usually denoted by* x, y, z*.*

From variables and function symbols in the first-order vocabularies, we can construct first-order terms, which are intuitively the objects we can model using first-order logic.

Definition 2.2 (First-order term)**.** *Given a first-order vocabulary* $(\mathsf{N_P}, \mathsf{N_F})$ *and a set of variables* $\mathsf{N_V}$*, the set of (first-order) terms is inductively defined as follows:*

(a) Variable $x \in \mathsf{N_V}$ *is a term;*

(b) Function terms *are of the form* $f(t_1, \ldots, t_n)$*, where* f *is a* n*-ary function symbol, and* $t_1, \ldots, t_n$ *are terms. Note that 0-ary function term* $f()$*, which is usually called constant, is normally abbreviated as* f *without parentheses.*

First-order formulas are built on the predicates and terms using connectives. They intuitively model the assertions about the objects and their relations.

Definition 2.3 (First-order formula)**.** *Given a first-order vocabulary* $(\mathsf{N_P}, \mathsf{N_F})$ *and a set of variables* $\mathsf{N_V}$*, the set of first-order formulas is inductively defined as follows:*

(a) Atomic formula*. If* p *is a* m*-ary predicate, and* $t_1, \ldots, t_n$ *are terms, then* $p(t_1, \ldots, t_m)$ *is a formula. When* $m = 0$*,* $p()$ *is usually abbreviated as* p*.*

(b) Negation*. If* α *is a formula, then* $\neg\alpha$ *is a formula.*

(c) Conjunction*. If* α_1 *and* α_2 *are formulas, then* $\alpha_1 \wedge \alpha_2$ *is a formula.*

(d) Disjunction*. If* α_1 *and* α_2 *are formulas, then* $\alpha_1 \vee \alpha_2$ *is a formula.*

(e) Implication*. If* α_1 *and* α_2 *are formulas, then* $\alpha_1 \to \alpha_2$ *is a formula.*

(f) Universal Quantification*. If* x *is a variable, and* α *is a formula, then* $\forall x\alpha$ *is a formula.*

(g) Existential Quantification*. If* x *is a variable, and* α *is a formula, then* $\exists x\alpha$ *is a formula.*

A *first-order theory* is a countable set of first-order formulas.

Propositional logic is a fragment of first-order logic, where predicates are restricted to be 0-ary predicates (also called propositional symbols), and without quantifications.

Sometimes, we assume that there is a "built-in" binary equality predicate $=$. In this case, we usually abbreviate the formula $=(x, y)$ as $x = y$, and $\neg =(x, y)$ as $x \neq y$.

Example 2.4. *Peano axioms of natural numbers are a first-order theory [Men97, Chap 3], which has a constant symbol* 0*, a unary function symbol* s*, and two binary function symbols* add *and* mul*, and the equality predicate* $=$*. For readability, terms* $add(t_1, t_2)$ *(resp.* $mul(t_1, t_2)$*) are written as* $t_1 + t_2$ *(resp.* $t_1 * t_2$*). Some of the axioms are as following:*

$$\forall x(x \neq s(x)) \tag{2.1}$$
$$\forall x \forall y(s(x) = s(y) \to x = y) \tag{2.2}$$
$$\forall x(x + 0 = x) \tag{2.3}$$
$$\forall x \forall y(x + s(y) = s(x + y)) \tag{2.4}$$
$$\forall x(x * 0 = 0) \tag{2.5}$$
$$\forall x \forall y(x * s(y) = x * y + x) \tag{2.6}$$

The intuition of the above theory is as follows: formulas (2.1) *and* (2.2) *are about the properties of the function* s *(which is for the successor of a natural number), formulas* (2.3) *and* (2.4) *inductively define the addition, and formulas* (2.5) *and* (2.6) *inductively define the multiplication.*

Definition 2.5 (Subformula)**.** *The subformulas of a formula* α *are* α *itself and all subformulas of immediate subformulas of* α*.*

- *Atomic formulas and $\top$ and $\bot$ have no immediate subformulas.*
- *The only immediate subformula of $\neg\alpha$ is α.*
- *The immediate subformulas of $(\alpha_1 \wedge \alpha_2)$ or $(\alpha_1 \vee \alpha_2)$ or $(\alpha_1 \to \alpha_2)$ are α_1 and α_2.*
- *The only immediate subformula of $\forall x\alpha$ or $\exists x\alpha$ is α.*

Definition 2.6 (Scope)**.** *Let α be a formula, Q a quantifier, and $Qx\beta$ a subformula of α. Then Qx is called a quantifier for x. Its scope in α is the subformula β except subformulas of β that begin with a quantifier for the same variable x.*

Each occurrence of x in the scope of Qx is bound in α by Qx. Each occurrence of x that is not in the scope of any quantifier for x is a free occurrence of x in α.

The formal meaning of the first-order logic is defined by the first-order semantics, which is given by interpretations and models.

Definition 2.7 (Variable assignment)**.** *Let D be a nonempty set. A variable assignment in D is a function V mapping each variable to an element of D. We denote the image of a variable x under an assignment V by x^V .*

Definition 2.8 (First-order Interpretation)**.** *Given a signature $(\mathsf{N_P}, \mathsf{N_F})$, an interpretation is a triple $\mathcal{I} = (D, I, V)$ where*

- *D is a nonempty set called the domain or universe (of discourse) of I.*
  Notation: $dom(\mathcal{I}) := D$.
- *I is a function defined on the symbols of $(\mathsf{N_P}, \mathsf{N_F})$ mapping*
  - *each n-ary function symbol $f \in \mathsf{N_F}$ to an n-ary function $f^I : D^n \to D$.*
    For $n = 0$ this means $f^I \in D$.
  - *each n-ary predicate symbol $p \in \mathsf{N_P}$ to an n-ary relation $p^I \subseteq D^n$.*
    For $n = 0$ this means either $p^I = \{\}$ or $p^I = \{()\}$.

  Notation: $f^{\mathcal{I}} := f^I$ and $p^{\mathcal{I}} := p^I$.
- *V is a variable assignment in D.*
  Notation: $x^{\mathcal{I}} := x^V$.

Notation 2.9. *Let V be a variable assignment in D, let V' be a partial function mapping variables to elements of D, which may or may not be a total function. Then $V[V']$ is the variable assignment with*

$$x^{V[V']} = \begin{cases} x^{V'} & \text{if } x^{V'} \text{ is defined,} \\ x^V & \text{otherwise.} \end{cases}$$

Let $\mathcal{I} = (D, I, V)$ be an interpretation. Then $\mathcal{I}[V'] := (D, I, V[V'])$.

By $\{x_1 \mapsto d_1, \ldots, x_k \mapsto d_k\}$ we denote the partial function that maps x_i to d_i and is undefined on other variables. In combination with the notation above, we omit the set braces and write $V[x_1 \mapsto d_1, \ldots, x_k \mapsto d_k]$ and $\mathcal{I}[x_1 \mapsto d_1, \ldots, x_k \mapsto d_k]$.

Definition 2.10 (Tarksi, model relationship)**.** *Let $\mathcal{I}$ be an interpretation and α a formula. The relationship $I \models \alpha$, pronounced "$\mathcal{I}$ is a model of α" or "$\mathcal{I}$ satisfies α" or "α is true in $\mathcal{I}$", and its negation $I \not\models \alpha$, pronounced "$\mathcal{I}$ falsifies α" or "α is false in $\mathcal{I}$", are defined inductively:*

- $\mathcal{I} \models p(t_1, \ldots, t_n)$, *if* $(t_1^{\mathcal{I}}, \ldots t_n^{\mathcal{I}}) \in p^{\mathcal{I}}$;
- $\mathcal{I} \models \top$;
- $\mathcal{I} \not\models \bot$;
- $\mathcal{I} \models \neg\alpha$, *if* $\mathcal{I} \not\models \alpha$;
- $\mathcal{I} \models \alpha_1 \wedge \alpha_2$, *if* $\mathcal{I} \models \alpha_1$ *and* $\mathcal{I} \models \alpha_2$;
- $\mathcal{I} \models \alpha_1 \vee \alpha_2$, *if* $\mathcal{I} \models \alpha_1$ *or* $\mathcal{I} \models \alpha_2$;
- $\mathcal{I} \models \exists x\alpha$, *if* $\mathcal{I}[x \mapsto d] \models \alpha$ *for at least one* $d \in D$;
- $\mathcal{I} \models \forall x\alpha$, *if* $\mathcal{I}[x \mapsto d] \models \alpha$ *for each* $d \in D$;

An interpretation $\mathcal{I}$ is a model of theory S, if $\mathcal{I}$ is a model of all formulas in S.

Based on the models, the following reasoning tasks are defined:

- *Satisfiability*. A formula α is satisfiable if there exists a model for α.
- *Validity*. A formula α is valid if $I \models \alpha$, for every interpretation I.
- *Entailment*. A formula α is entailed by a theory S, denoted $S \models \alpha$, if $I \models \alpha$ for all the models of the S.

These reasoning tasks can be reduced to each other. Let α be a formula, then

(1) α is satisfiable iff $\neg\alpha$ is not valid;

(2) α is unsatisfiable iff $\{\alpha\} \models \bot$;

(3) α is valid iff $\neg\alpha$ is not satisfiable;

(4) for every set S of formulas, $S \models \alpha$ iff $S \wedge \neg\alpha$ is not satisfiable.

Example 2.11. *We show an interpretation $\mathcal{I}$ for formulas* (2.1) *to* (2.6) *of Peano axioms in Example 2.4.*

- *Domain* $D = \{\underbrace{1\ldots1}_{n} \mid n \geq 0\}$. *We denote the element when* $n = 0$ *by* ϵ.
- *Constant* $0^{\mathcal{I}} = \epsilon$.
- *Functions* $s^{\mathcal{I}}(\underbrace{1\ldots1}_{m}) = \underbrace{1\ldots1}_{m+1}$, $\quad mul^{\mathcal{I}}(\underbrace{1\ldots1}_{m}, \underbrace{1\ldots1}_{n}) = \underbrace{1\ldots1}_{mn}$, $\quad add^{\mathcal{I}}(\underbrace{1\ldots1}_{m}, \underbrace{1\ldots1}_{n}) = \underbrace{1\ldots1}_{m+n}$

- *Predicate* $(=^{\mathcal{I}}) = \{(\underbrace{1...1}_{n}, \underbrace{1...1}_{n}) \mid n \geq 0\}$.

Indeed, the intuition of this interpretation is the unary encoding of natural numbers. It is easy to verify that $\mathcal{I}$ is a model of formulas (2.1) *to* (2.6)*.*

Finally we remark that first-order logic is very expressive, however it is not well-suited as a practical knowledge representation language because of its high complexity. As we will see the in the next section, first-order logic is even undecidable in general.

2.2 Computational Complexity

In this section, we recall the basic notions and results of computational complexity theory that will be used in this thesis. The content of this section is based on the teaching materials used in the lecture of *Formal Methods in Computer Science*[1] and *Complexity Theory*[2], by Reinhard Pichler, in Vienna University of Technology. For more details about the complexity theory, there are several excellent textbooks, e.g. the one by Christos H. Papadimitriou [Pap94].

Complexity theory focuses on analyzing the computational complexity of problems (not algorithms). Here a problem is an infinite set of possible instances with a question. We are particularly interested in the decision problems whose questions have only yes/no answers. For example, in the problem of satisfiability of the first-order logic, the instances are all first-order formulas, and the question is whether they are satisfiable.

2.2.1 Turing Machine

Turing Machine is a computational model which is widely accepted to be a universal model for computation.

Definition 2.12. *A deterministic Turing machine (DTM) is a quadruple $M = (S, \Sigma, \delta, s)$ with a finite set of states S, a finite set of symbols Σ (alphabet of M) so that $\textvisiblespace, \triangleright \in \Sigma$, a transition function δ:*

$$S \times \Sigma \to (S \cup \{\textbf{\textit{yes}}, \textbf{\textit{no}}\}) \times \Sigma \times \{-1, 0, 1\},$$

*an accepting state **yes**, a rejecting state **no**, and cursor directions: -1 (left), 0 (stay), and $+1$ (right).*

Let M be a DTM (S, Σ, δ, s_0). The tape of M is divided into cells containing symbols of Σ. There is a cursor that may move along the tape. At the start, M is in the initial state s_0, and the cursor points to the leftmost cell of the tape. An input string I is written on the tape as follows: the first $|I|$ cells $c_0, \ldots, c_{|I|-1}$ of the tape, where $|I|$ denotes the length of I, contains the symbols of I, and all other cells contain $\textvisiblespace$.

The machine takes successive steps of computation according to δ. Namely, assume that M is in a state $s \in S$ and the cursor points to the symbol $\sigma \in \Sigma$ on the tape. Let $\delta(s, \sigma) = (s', \sigma', d)$.

[1] Course website: http://www.logic.at/lvas/185291/

[2] Course website: http://www.dbai.tuwien.ac.at/staff/pichler/complexity/index.html

Then M changes its current state to s', overwrites σ' on σ, and moves the cursor according to d. Namely, if $d = -1$ or $d = +1$, then the cursor moves to the previous cell or the next one, respectively; if $d = 0$, then the cursor remains in the same position. When any of the states **yes** or **no** is reached, M halts. We say that M accepts the input I if M halts in **yes**. Similarly, we say that M rejects the input in the case of halting in **no**.

Solving a decision problem amounts to deciding the language consisting of the encodings of the **yes** instances of the problem. For instance, the language for the problem of satisfiability of first-order logic is the set of all the satisfiable formulas. Given a set of symbols, a (formal) language L is an (infinite) set of strings of symbols.

Definition 2.13. *Let $L \subseteq (\Sigma - \{\triangleright, \sqcup\})^*$ be a language. A Turing machine M decides L iff for every string $x \in (\Sigma - \{\triangleright, \sqcup\})^*$, the following conditions hold: if $x \in L$, $M(x) =$ "yes" and if $x \notin L, M(x) =$ "no". A language L is decidable, if L is decided by a Turing Machine; otherwise, it is undecidable.*

It is easy to see that the number of Turing machines is countable, so they can decide only countably many languages. Since there are uncountably many languages, there must exist undecidable languages. One famous undecidable language is first-order logic.

Theorem 2.14. *The problem of deciding the satisfiability of formulas in first-order logic is undecidable.*

In the definition of Turing machine, the computation is deterministic, as the transition of status is a function. A variant of Turning machine is non-deterministic: at each step, there can be many different possibilities for the next step.

Definition 2.15. *A non-deterministic Turing machine (NTM) is a quadruple $N = (K, \Sigma, \Delta, s)$ like the ordinary Turing machine except that Δ is a transition relation (rather than a transition function):*

$$\Delta \subseteq (K \times \Sigma) \times [(K \times \{\textbf{\textit{yes}}, \textbf{\textit{no}}\}) \times \Sigma \times \{-1, 0, +1\}]$$

A tuple of a transition relation whose first two members are s and σ respectively, specifies the action of the NTM when its current state is s and the symbol pointed at by its cursor is σ. If the number of such tuples is greater than one, the NTM non-deterministically chooses any of them and operates accordingly.

Unlike the case of a DTM, the definition of acceptance and rejection by an NTM is asymmetric. We say that an NTM accepts an input if there is at least one sequence of choices leading to the state **yes**; an NTM rejects an input if no sequence of choices can lead to **yes**.

2.2.2 Complexity Classes

Definition 2.16. *A deterministic Turing machine M decides a language L in time $f(n)$ iff M decides L and for any $x \in \Sigma^*$ and k is the steps of deciding x by M, then $k \leq f(|x|)$.*

Definition 2.17. *A non-deterministic Turing machine N decides a language L in time $f(n)$ iff N decides L and for any $x \in \Sigma^*$ and k is the steps of deciding x by N, then $k \leq f(|x|)$.*

Definition 2.18. *A time complexity class* $\text{TIME}(f(n))$ *(*$\text{NTIME}(f(n))$*) is the set of languages* L *such that* L *is decided by a deterministic (non-deterministic) Turing machine in time* $O(f(n))$*.*

Definition 2.19. *A space complexity class* $\text{SPACE}(f(n))$ *(*$\text{NSPACE}(f(n))$*) is the set of languages* L *such that* L *is decided by a deterministic (non-deterministic) Turing machine within space* $O(f(n))$*.*

Definition 2.20. *Some of the important complexity classes are as follows:*

$$\begin{aligned}
\mathsf{P} &= \bigcup_{k\in\mathbb{N}} (\text{TIME}(O(n^k))) \\
\mathsf{NP} &= \bigcup_{k\in\mathbb{N}} (\text{NTIME}(O(n^k))) \\
\mathsf{PSPACE} &= \bigcup_{k\in\mathbb{N}} (\text{SPACE}(O(n^k))) \\
\mathsf{NPSPACE} &= \bigcup_{k\in\mathbb{N}} (\text{NSPACE}(O(n^k))) \\
\mathsf{EXPTIME} &= \bigcup_{k\in\mathbb{N}} (\text{TIME}(O(2^{n^k}))) \\
\mathsf{NEXPTIME} &= \bigcup_{k\in\mathbb{N}} (\text{NTIME}(O(2^{n^k}))) \\
\mathsf{2EXPTIME} &= \bigcup_{k\in\mathbb{N}} (\text{TIME}(O(2^{2^{n^k}}))) \\
\mathsf{N2EXPTIME} &= \bigcup_{k\in\mathbb{N}} (\text{NTIME}(O(2^{2^{n^k}})))
\end{aligned}$$

Any complexity class $\mathcal{C}$ has its complementary class denoted by co-$\mathcal{C}$ and defined as follows.

Definition 2.21. *For every language* L *in* Σ^**, let* L *denote its complement i.e.* $\overline{L} = \Sigma^* \setminus L$*. Then co-*$\mathcal{C}$ *is defined as co-*$\mathcal{C} = \{\overline{L} \mid L \in \mathcal{C}\}$*.*

2.2.3 Reductions

To classify languages to computational complexity classes, we need the notion of reduction.

Definition 2.22. *Let* L_1 *and* L_2 *be decision problems (i.e., languages over some alphabet* Σ*). A reduction from language* L *to* L' *is a function* $f : \Sigma^* \to \Sigma^*$ *, s.t.* $x \in L$ *iff* $f(x) \in L'$*. We are interested in reduction can be computed computed efficiently, e.g. in polynomial time or logarithm space. The language* L *can be (polynomially) reduced to* L'*, denoted* $L \leq L'$*, if there exists a (polynomial) reduction. In the following of this thesis, we assume all the reductions are polynomial, unless otherwise stated.*

Definition 2.23. *Let $\mathcal{C}$ be a complexity class and let L be a language in $\mathcal{C}$. Then L is $\mathcal{C}$-complete if for every $L' \in \mathcal{C}$, $L' \leq L$. A language L is called $\mathcal{C}$-hard if any language $L' \in \mathcal{C}$ is reducible to L.*

The complexity classes in Definition 2.20 form the following hierarchy:

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{NEXPTIME} \subseteq \mathsf{2EXPTIME} \subseteq \mathsf{N2EXPTIME}$$

In the bottom of this hierarchy, it is the important class P. It is generally accepted that P is the *tractable* class and the problems in P can be solved efficiently. In contrast, NP is believed to be *intractable* and the problems in NP (and beyond NP) can not be solved very efficiently in theory. The problem that whether $\mathsf{P} = \mathsf{NP}$ holds is a very important open issue in computer science.

2.2.4 Oracle Turing Machine and Polynomial Hierarchy

A Turing Machine can invoke another Turing machine, called Oracle Turing Machine, as a sub module. For any time complexity class $\mathcal{C}$ and oracle A (where A is either a problem or a class of problems) we write $\mathcal{C}^A$ for the problems which can be decided by a TM within the time bound of $\mathcal{C}$, where the TM is allowed to use an oracle for (any problem in the class) A.

Definition 2.24. *The polynomial hierarchy is a sequence of classes:*

- $\Delta_0^{\mathsf{P}} = \Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$,
- *For* $i \geq 0$*:* $\Delta_{i+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_i^{\mathsf{P}}}, \Sigma_{i+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}, \Pi_{i+1}^{\mathsf{P}} = \textit{co}\text{-}\mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$,
- *Cumulative polynomial hierarchy:* $\mathsf{PH} = \bigcup_{i \geq 0} \Sigma_i^{\mathsf{p}}$.

The classes in polynomial hierarchy have the following relation.

$$\mathsf{P} \subseteq \begin{matrix} \Sigma_1^{\mathsf{P}} (= \mathsf{NP}) \\ \Pi_1^{\mathsf{P}} (= \text{co-}\mathsf{NP}) \end{matrix} \subseteq \Delta_2^{\mathsf{P}} \subseteq \begin{matrix} \Sigma_2^{\mathsf{P}} \\ \Pi_2^{\mathsf{P}} \end{matrix} \subseteq \Delta_3^{\mathsf{P}} \subseteq \begin{matrix} \Sigma_3^{\mathsf{P}} \\ \Pi_3^{\mathsf{P}} \end{matrix} \subseteq \cdots \subseteq \mathsf{PH}.$$

2.3 Description Logics and OWL

Description Logics (DLs) are a family of KR formalisms underpinning the Web Ontology Language (OWL). DLs are decidable, but yet expressive fragments of first-order logic. The syntax of DLs only uses predicate names and constants, but not variables, which can be seen as a syntax sugar for the corresponding fragment of first-order logics.

2.3.1 Expressive Description Logics

We start with the "basic" description logic $\mathcal{ALC}$ (Attributive Language with Complement).

Definition 2.25 (DL Vocabulary)**.** *A DL vocabulary $\mathcal{V}$ is a triple* $(\mathsf{N_C}, \mathsf{N_R}, \mathsf{N_I})$ *where* $\mathsf{N_C}$ *is a set of* concept names*,* $\mathsf{N_R}$ *a set of* role names*, and* $\mathsf{N_I}$ *a set of* individual names*.*

Constructor	description
$\top$	top concept
$\bot$	bottom concept
A	concept names
$C_1 \sqcap C_2$	conjunction
$C_2 \sqcup C_2$	disjunction
$\neg C_1$	negation (complement)
$\forall R.C_1$	universal quantification
$\exists R.C_1$	existential quantification

Table 2.1: concept expressions in $\mathcal{ALC}$

In the literature, concept (role) names are also known as *atom concepts (roles)*. In OWL 2 specification, concepts and roles are called *classes* and *properties* respectively.

In $\mathcal{ALC}$, concept expressions can be built by *conjunction*, *disjunction*, *negation*, *universal quantification*, and *existential quantification*. Formally, the constructors in $\mathcal{ALC}$ are listed in Table 2.1.

Intuitively, individuals are objects or instances; concepts are unary predicates about the objects; and roles are binary predicates modeling the relations between objects. The concepts are either (1) concept names or (2) complex concept expressions.

Note that $\forall R.\top$ (resp. $\exists R.\top$) can be abbreviated to $\forall R$ (resp. $\exists R$).

Example 2.26. *Let* $\mathsf{N_C} = \{Male, Female, Person, Student, Professor\}, \mathsf{N_R} = \{hasChild\}$. *For instance, we can build the following concepts:*

Concept	*Intended meaning*
$Male \sqcap Person$	*male person*
$Student \sqcup Professor$	*student or professor*
$Student \sqcap (\neg Male)$	*non-male student*
$Male \sqcap Person \sqcap \exists hasChild.Person$	*male person who has a child which is a person*
$Female \sqcap Person \sqcap \forall hasChild.Female$	*Female person whose children are all female*

DL $\mathcal{ALC}$ allows us to express the assertions about individuals and relations between concepts in the following forms:

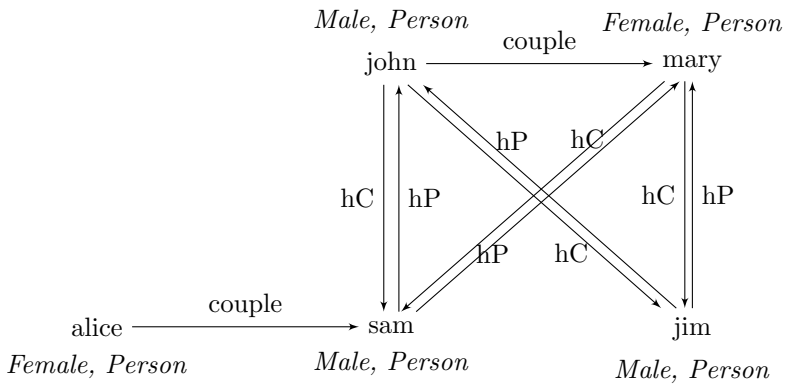
(1) $C(a)$: a is a instance of C,

(2) $R(a, b)$: a and b are related by R,

(3) $C \sqsubseteq D$: C is a subconcept of D,

(4) $C \equiv D$: C is equivalent to D.

Axioms of the form (1) and (2) are called *assertion axioms*; axioms of the form (3) and (4) are called *terminological axioms*. A set of assertion axioms is called *assertion box (ABox)*, and a set of terminological axioms is called *terminological box (TBox)*. An ontology $\mathcal{O}$ is a pair



hC=hasChild, hP=hasParent

Figure 2.1: Family Tree

$(\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is a ABox. We say $\mathcal{O}$ is in $\mathcal{ALC}$ if all the axioms are $\mathcal{ALC}$ axioms.

Example 2.27. *The following ontology $\mathcal{O}_f = (\mathcal{T}_f, \mathcal{A}_f)$ is an $\mathcal{ALC}$ ontology for family relation.*

$$\mathcal{T}_f = \left\{ \begin{array}{r} Father \equiv Male \sqcap Person \sqcap \exists hasChild.Person, \\ Mother \equiv Female \sqcap Person \sqcap \exists hasChild.Person, \\ Son \equiv Male \sqcap Person \sqcap \exists hasParent.Person, \\ Daughter \equiv Female \sqcap Person \sqcap \exists hasParent.Person, \\ Person \sqsubseteq \exists hasParent.(Male \sqcap Person) \\ Person \sqsubseteq \exists hasParent.(Female \sqcap Person) \\ Male \sqsubseteq \neg Female \\ Person \sqsubseteq \forall hasChild.Person \end{array} \right\}$$

$$\mathcal{A}_f = \left\{ \begin{array}{llll} Person(john). & Person(mary). & Person(alice). & Person(sam). \quad Person(jim). \\ Male(john). & Female(mary). & Female(alice). & Male(sam). \quad Male(jim). \\ hasChild(john, sam). & hasChild(john, jim). & hasChild(mary, sam). & \\ hasChild(mary, jim). & couple(john, mary). & couple(alice, sam) & \end{array} \right\}$$

Semantics

The semantics of $\mathcal{ALC}$ is the standard first-order semantics defined by interpretations $\mathcal{I}$.

Definition 2.28. *An interpretation $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is set called domain, and $\cdot^{\mathcal{I}}$ is a function from $\mathsf{N_I}$ to $\Delta^{\mathcal{I}}$, such that:*

- $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$;
- $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$;
- $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$;
- $(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$;

	Mapping Concepts to FOL	
$\pi_y(\top, X)$	=	$\top$
$\pi_y(\bot, X)$	=	$\bot$
$\pi_y(A, X)$	=	$A(X)$
$\pi_y(\neg C, X)$	=	$\neg \pi_y(C, X)$
$\pi_y(C \sqcap D, X)$	=	$\pi_y(C, X) \wedge \pi_y(D, X)$
$\pi_y(C \sqcup D, X)$	=	$\pi_y(C, X) \vee \pi_y(D, X)$
$\pi_y(\forall R.C, X)$	=	$\forall y : R(X, y) \rightarrow \pi_x(C, y)$
$\pi_y(\exists R.C, X)$	=	$\exists y : R(X, y) \wedge \pi_x(C, y)$
$\pi_y(\leq nR.C, X)$	=	$\forall y_1, \ldots, y_{n+1} : \bigwedge_{i=1}^{n+1} [R(X, y_i) \wedge \pi_x(C, y_i)] \rightarrow \bigvee_{i=1}^{n+1} \bigvee_{j=i+1}^{n+1} y_i \not\approx y_j$
$\pi_y(\geq nR.C, X)$	=	$\exists y_1, \ldots, y_n : \bigwedge_{i=1}^{n+1} [R(X, y_i) \wedge \pi_x(C, y_i)] \wedge \bigwedge_{i=1}^{n+1} \bigwedge_{j=i+1}^{n+1} y_i \not\approx y_j$
$\pi_x(\{a\})$	=	$(x \approx a)$
$\exists S.Self$	=	$S(x, x)$
	Mapping Roles to FOL	
$\pi_{x,y}(S)$	=	$S(x, y)$
$\pi_{x,y}(R^-)$	=	$\pi_{y,x}(R)$
$\pi_{x,y}(R_1 \circ \cdots \circ R_n)$	=	$\exists x_1, \ldots, x_{n-1}\ (\pi_{x,x_1}(R_1)) \wedge \bigwedge_{i=1}^{n-2} \pi_{x_i,x_{i+1}}(R_{i+1}) \wedge \pi_{x_{n-1},y}(R_n)$
	Mapping Axioms to FOL	
$\pi(C \sqsubseteq D)$	=	$\forall x : \pi_y(C, x) \rightarrow \pi_y(D, x)$
$\pi(C \equiv D)$	=	$\forall x : \pi_y(C, x) \leftrightarrow \pi_y(D, x)$
$\pi(R \sqsubseteq S)$	=	$\forall x : \pi_{x,y}(R) \rightarrow \pi_{x,y}(S)$
$\pi(\mathsf{Trans}(R))$	=	$\forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z)$
$\pi(C(a))$	=	$\pi_y(C, a)$
$\pi(R(a, b))$	=	$R(a, b)$
$\pi(\neg S(a, b))$	=	$\neg S(a, b)$
$\pi(a \circ b)$	=	$a \circ b$ for $\circ \in \{\approx, \not\approx\}$
$\pi(\mathsf{Ref}(R))$	=	$(\forall x)\pi_{x,x}(R)$
$\pi(\mathsf{Asy}(R))$	=	$(\forall x)(\forall y)\pi_{x,y}(R) \rightarrow \neg \pi_{y,x}(R)$
$\pi(\mathsf{Dis}(R_1, R_2))$	=	$\neg(\exists x)(\exists y)(\pi_{x,y}(R_1) \wedge \pi_{x,y}(R_2))$
	Mapping Ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A}, \mathcal{R})$ to FOL	
$\pi(R)$	=	$\forall x, y : R(x, y) \leftrightarrow R^-(y, x)$
$\pi(\mathcal{R})$	=	$\bigwedge_{\alpha \in \mathcal{R}} \wedge \bigwedge_{R \in \mathsf{N_R}} \pi(R)$
$\pi(\mathcal{T})$	=	$\bigwedge_{\alpha \in \mathcal{T}}$
$\pi(\mathcal{A})$	=	$\bigwedge_{\alpha \in \mathcal{A}}$
$\pi(\mathcal{O})$	=	$\pi(\mathcal{R}) \wedge \pi(\mathcal{T}) \wedge \pi(\mathcal{A})$

Table 2.2: Semantics of $\mathcal{SROIQ}$ by Mapping to FOL

Notes:

(1) X is a meta-variable and is substituted by the actual term;

(2) π_x is obtained from π_y by simultaneously substituting in the definition all $y_{(i)}$ with $x_{(i)}$, π_y with π_x, and vice versa.

- $(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$;
- $(\neg C_1)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C_1^{\mathcal{I}}$;
- $(\exists R.C)^{\mathcal{I}} = \{x \mid$ *there exists* $y_0 \in \Delta^{\mathcal{I}}$, *such that* $(x, y_0) \in R^{\mathcal{I}}$ *and* $y_0 \in C^{\mathcal{I}}\}$;
- $(\forall R.C)^{\mathcal{I}} = \{x \mid$ *if* $(x, y) \in R^{\mathcal{I}}$ *then* $y \in C^{\mathcal{I}}\}$.

The satisfiability of an axiom α w.r.t an interpretation $\mathcal{I}$ *is defined as follows:*

- $\mathcal{I} \models C(a)$, *if* $a^{\mathcal{I}} \in C^{\mathcal{I}}$;
- $\mathcal{I} \models R(a, b)$, *if* $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$;
- $\mathcal{I} \models C \sqsubseteq D$, *if* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$;
- $\mathcal{I} \models C \equiv D$, *if* $C^{\mathcal{I}} = D^{\mathcal{I}}$.

An axiom (resp. TBox, ABox, ontology) is satisfiable (or consistent) if it is satisfiabile w.r.t some interpretation $\mathcal{I}$.

Note that often Description logics are said to be fragments of first-order logic with equality. There is a translation of $\mathcal{SROIQ}$ (logic underpinning OWL 2 DL) into first-order logic shown in Table 2.2 which is equivalent to the above definition.

Example 2.29. $\mathcal{I}_{f_1} = (\Delta^{\mathcal{I}_{f_1}}, \cdot^{\mathcal{I}_{f_1}})$ *is an interpretation for ontology* O_f *in Example 2.26.*

The domain is natural numbers: $\Delta^{\mathcal{I}_{f_1}} = \mathbb{N}$.

Interpretation of individuals:

$$john^{\mathcal{I}_{f_1}} = 1 \qquad mary^{\mathcal{I}_{f_1}} = 2 \qquad alice^{\mathcal{I}_{f_1}} = 3 \qquad sam^{\mathcal{I}_{f_1}} = 4 \qquad jim^{\mathcal{I}_{f_1}} = 5$$

Interpretation of concepts:

$$\begin{aligned}
Person^{\mathcal{I}_{f_1}} &= \{1,2,3,4,5\} \cup \{11,12,111,112,121,122,1111,1112,\ldots,\} \\
&\quad \cup \{21,22,211,212,221,222,\ldots\} \\
&\quad \cup \{31,32,311,312,321,322,\ldots\} \\
Male^{\mathcal{I}_{f_1}} &= \{1,4,5\} \cup \{11,111,121,1111,1121,\ldots\} \\
&\quad \cup \{21,211,221,2111,2121,\ldots\} \\
&\quad \cup \{31,311,321,3111,3121,\ldots\} \\
Female^{\mathcal{I}_{f_1}} &= \{2,3\} \cup \{12,112,122,1112,1122,\ldots\} \\
&\quad \cup \{22,212,222,2112,2122,\ldots\} \\
&\quad \cup \{32,312,322,3112,3122,\ldots\} \\
Son^{\mathcal{I}_{f_1}} &= Male^{\mathcal{I}_{f_1}} \\
Daughter^{\mathcal{I}_{f_1}} &= Female^{\mathcal{I}_{f_1}} \\
Father^{\mathcal{I}_{f_1}} &= \{1\} \\
Mother^{\mathcal{I}_{f_1}} &= \{2\}
\end{aligned}$$

Interpretation of roles:

$$\begin{aligned} hasChild^{\mathcal{I}_{f_1}} &= \{(1,4),(2,4),(1,5),(2,5)\} \\ hasParent^{\mathcal{I}_{f_1}} &= \{(4,1),(4,2),(5,1),(5,2)\} \\ &\cup \{(1,11),(1,12),(12,121),(12,122),\ldots\} \\ &\cup \{(2,21),(2,22),(22,221),(22,222),\ldots\} \\ &\cup \{(3,31),(3,32),(32,321),(32,322),\ldots\} \end{aligned}$$

Logical entailment an axiom α is a *logical entailment* of an ontology O, if for all the interpretations of O are also interpretations of α. In other words, models of α are a superset of models of O.

Beyond the basic constructors in $\mathcal{ALC}$, there are more constructors for exprssive description logics. We list all the concept constructors and axioms of $\mathcal{SROIQ}$ in Table 2.3. Note that these features are not completely independent. For instance, unqualified number restriction $\mathcal{N}$ is a special case of qualified number restriction $\mathcal{Q}$. These constructors can be combined to form fragments of description logics. By convention, $\mathcal{ALC}$ with transitivity($^+$) is abbreviated as $\mathcal{S}$. Some common combinations are: $\mathcal{SHIF}$ (logic underpinning OWL Lite), $\mathcal{SHOIN}$ (underpinning OWL 1), $\mathcal{SROIQ}$ (underpinning OWL 2).

For decidability, some syntax restrictions have to be applied on the expressive description logics.

(1) In $\mathcal{SHIQ}$ (and all DLs containing $\mathcal{SHIQ}$), in any qualified number restriction $\geq nR.C$ and $\leq nR.C$, role R has to be restricted to simple roles, that is R cannot have transitive sub roles.

(2) In $\mathcal{SROIQ}$, complex role inclusion axioms are restricted to the regular ones [HKS06]. Regularity prevents a role hierarchy from containing cyclic dependencies that may lead to undecidablity.

Reasoning Tasks

Now we are ready to talk about some *reasoning tasks* in $\mathcal{ALC}$ (and in general DLs).

- *Satisfiability* (or *consistency*). Given an ontology L, determine whether there exists an interpretation for L.
- *Ground instance query*. Given an ontology L and an assertion $C(a)$, determine whether $L \models C(a)$ holds.
- *Instance query*. Given an ontogeny L and a concept C, compute all the instances x such that $L \models C(x)$.
- *Subsumption checking*. Given an ontology L and an assertion $C \sqsubseteq D$, determine whether $L \models C \sqsubseteq D$ holds.

- *Classification*. Given an ontology L, compute all pairs of *atomic* concepts (A, B) such that $L \models A \sqsubseteq B$ holds.

Note that all of the above reasoning tasks can be reduced to the (un)satisfiability [HP04a].

Example 2.30. *Reconsider the ontology O_f in Example 2.26. Clearly, O_f is satisfiable. It is easy to check that $O_f \models Father \sqsubseteq Son$, $O_f \models Son \sqsubseteq Male$, $O_f \models Son(sam)$, $O_f \models Son(john)$, and $O_f \models Father(john)$ hold.*

Sometimes we are interested in the interpretations which assign different meanings to different individuals. Formally, an interpretation $\mathcal{I}$ is under unique name assumption (UNA), if $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ holds for all $a \neq b \in \mathsf{N_I}$.

Queries over Description Logics A query is an open formula of first-order logic (FOL) with equalities. Formally, a FOL query q is an expression of the form

$$\{\, \mathbf{X} \mid \phi(\mathbf{X}) \,\},$$

where $\phi(\mathbf{X})$ is a FOL formula with free variables $\mathbf{X}$. We call the size of $\mathbf{X}$ the arity of the query q. Given an interpretation $\mathcal{I}$, $q^{\mathcal{I}}$ is the set of tuples of domain elements that, when assigned to the free variables, make the formula ϕ true in $\mathcal{I}$.

A conjunctive query (CQ) q is a query of the form

$$\{\mathbf{X} \mid \exists \mathbf{Y}.conj(\mathbf{X}, \mathbf{Y})\},$$

where $conj(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms and equalities, with free variables $\mathbf{X}$ and $\mathbf{Y}$. Sometimes, we use the standard Datalog notation

$$q(\mathbf{X}) \leftarrow conj(\mathbf{X}, \mathbf{Y}).$$

Similarly, a union of conjunctive queries (UCQ) q is a query of the form

$$\left\{ \mathbf{X} \mid \bigvee_{i=1,\ldots,n} \exists \mathbf{Y}_i.conj_i(\mathbf{X}, \mathbf{Y}_i) \right\},$$

where each $conj_i(\mathbf{X}, \mathbf{Y}_i)$ is, as before, a conjunction of atoms and equalities with free variables $\mathbf{X}$ and $\mathbf{Y}_i$. A UCQ q can also be expressed in the standard Datalog notation as

$$\begin{aligned} q(\mathbf{X}) &\leftarrow conj_1(\mathbf{X}, \mathbf{Y}_1). \\ q(\mathbf{X}) &\leftarrow conj_2(\mathbf{X}, \mathbf{Y}_2). \\ &\cdots \\ q(\mathbf{X}) &\leftarrow conj_n(\mathbf{X}, \mathbf{Y}_n). \end{aligned}$$

Given a query q (either a CQ or a UCQ) and an ontology L, the answer to q over L is the set $ans(q, L)$ of tuples $\mathbf{a}$ of constants appearing in L such that $\mathbf{a}^{\mathcal{I}} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of L.

	Syntax	Semantics
Concept constructors		
Top	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
Bottom	$\bot$	$\bot^{\mathcal{I}} = \emptyset$
Atomic concepts	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Negation ($\mathcal{C}$)	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Existential quant.	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \text{ there is } y_0 \in \Delta^{\mathcal{I}}, \text{ s.t. } (x, y_0) \in R^{\mathcal{I}} \wedge y_0 \in C^{\mathcal{I}}\}$
Universal quant.	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \text{ if } (x, y) \in R^{\mathcal{I}} \text{ then } y \in C^{\mathcal{I}}\}$
Unqualified number restrictions ($\mathcal{N}$)	$\leq nR.\top$ $\geq nR.\top$	$(\leq nR.\top)^{\mathcal{I}} = \{x \mid \#\{x \mid (x, y) \in R^{\mathcal{I}}\} \leq n\}$ $(\geq nR.\top)^{\mathcal{I}} = \{x \mid \#\{x \mid (x, y) \in R^{\mathcal{I}}\} \geq n\}$
Functionality ($\mathcal{F}$)	$\leq 1.R$	$(\leq 1R)^{\mathcal{I}} = \{x \mid \#\{x \mid (x, y) \in R^{\mathcal{I}}\} \leq 1\}$
Qualified number restrictions ($\mathcal{Q}$)	$\leq nR.C$ $\geq nR.C$	$(\leq nR.C)^{\mathcal{I}} = \{x \mid \#\{x \mid (x, y) \in R^{\mathcal{I}}, y \in C^{\mathcal{I}}\} \leq n\}$ $(\geq nR.C)^{\mathcal{I}} = \{x \mid \#\{x \mid (x, y) \in R^{\mathcal{I}}, y \in C^{\mathcal{I}}\} \geq n\}$
Nominals ($\mathcal{O}$)	$\{o_1, \ldots, o_k\}$	$\{o_1, \ldots, o_k\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \ldots, o_k^{\mathcal{I}}\}$
Self	$\exists R.\mathsf{Self}$	$(\exists R.\mathsf{Self})^{\mathcal{I}} = \{x \mid (x, x) \in R^{\mathcal{I}}\}$
Role Constructors		
Atomic roles	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Inverse roles ($\mathcal{I}$)	R^-	$(R^-)^{\mathcal{I}} = \{(x, y) \mid (y, x) \in R^{\mathcal{I}}\}$
ABox Axioms		
Concept assertions	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
Role assertions	$R(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
TBox Axioms		
Concept subsumption	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
RBox Axioms		
Role hierarchy ($\mathcal{H}$)	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
Role transitivity ($^+$)	$trans(R)$	$R^{\mathcal{I}} \circ R^{\mathcal{I}} = R^{\mathcal{I}}$
Complex role inclusion ($\mathcal{R}$)	$R \circ S \sqsubseteq R$ $S \circ R \sqsubseteq R$	$R^{\mathcal{I}} \circ S^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ $S^{\mathcal{I}} \circ R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

Table 2.3: Syntax and semantics of Description Logic $\mathcal{SROIQ}$

Language	Combined complexity	Data complexity
$\mathcal{ALC}$	EXPTIME-complete	NP-complete
$\mathcal{SHIF}$	EXPTIME-complete	NP-complete
$\mathcal{SHIQ}$	EXPTIME-complete	NP-complete
$\mathcal{SHOIN}$	NEXPTIME-complete	NP-hard
$\mathcal{SROIQ}$	N2EXPTIME-complete	NP-hard

Table 2.4: Computational Complexity of Expressive DLs

Complexities

The computational complexity of DLs are well studied. There are two settings used for complexity in description logics: (1) combined complexity, which takes all the components in the ontology as inputs, and (2) data complexity, which fixes the TBox and RBox and only considers ABox as inputs. We summarize some complexity results of satisfiability checking from [Kaz08] in Table 2.4. For more results, Evgeny Zolin maintains an extensive list for reference[3].

2.3.2 Lightweight Description Logics

We have seen that reasoning in expressive DLs is intractable (Table 2.4). To tackle the high complexity, apart from optimizing the reasoning algorithm for expressive DLs, researchers also identified lightweight fragments of DLs which have low complexity and are still expressive enough in many applications. These lightweight logics include DL-Lite family (underpinning OWL 2 QL), $\mathcal{EL}$ family (underpinning OWL 2 EL), and DLP (underpinning OWL 2 RL). For all these OWL 2 profiles, see [Mot+12].

DL-Lite family and OWL 2 QL

Consider a vocabulary of individual names $\mathsf{N_I}$, *atomic concepts* $\mathsf{N_C}$, and *atomic roles* $\mathsf{N_R}$. Then, for A and P being an atomic concept and atomic role, respectively, we define *basic* concepts B and *basic* roles R, *complex* concepts C and *complex* role expressions E as

$$\begin{array}{rcl|rcl} B & ::= & A \mid \exists R & C & ::= & B \mid \neg B \\ R & ::= & P \mid P^- & E & ::= & R \mid \neg R \end{array}$$

where P^- is the inverse of P.

In DL-Lite$_{core}$, TBox $\mathcal{T}$ consists of a finite set of *inclusion assertions* of the form $B \sqsubseteq C$. DL-Lite^R additionally allow $R \sqsubseteq E$. As usual, a DL-Lite ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ is pair of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$.

The main reasoning task in DL-Lite is conjunctive query answering. Answering CQ over DL-Lite ontologies can be done by query rewriting [Cal+07]: given a DL-Lite TBox $\mathcal{T}$, the CQ q over $\mathcal{T}$ can be rewritten to a UCQ $Q_\mathcal{T}$, such that $ans(q, (\mathcal{T}, \mathcal{A})) = ans(Q_\mathcal{T}, \mathcal{A})$ for any ABox $\mathcal{A}$. Recall that in plain database, conjunctive query is AC^0 complete under data complexity [AHV95]. Therefore, DL-Lite also enjoys this attractive low data complexity for conjunctive queries.

$\mathcal{EL}$ family and OWL 2 EL

According to W3C OWL 2 Profiles [Mot+08], the OWL 2 EL profile is designed as a subset of OWL 2 that is particularly suitable for applications employing ontologies that define very large numbers of classes and/or properties, captures the expressive power used by many such ontologies, and for which the reasoning tasks can be done in polynomial time.

[3] http://www.cs.man.ac.uk/~ezolin/dl/

OWL 2 EL is based on the family of $\mathcal{EL}$ logics, which includes $\mathcal{EL}$, $\mathcal{EL}^+$, $\mathcal{EL}^{++}$ and $\mathcal{SROEL}(\sqcap, \times)$ [BBL05; BBL08; Krö11]. We recall $\mathcal{SROEL}(\sqcap, \times)$ as it is the most expressive logic in the $\mathcal{EL}$ family.

Consider a vocabulary $(\mathsf{N_I}, \mathsf{N_C}, \mathsf{N_R})$, for A and P being an atomic concept and atomic role, respectively, the set $\mathbf{C}$ of $\mathcal{SROEL}(\sqcap, \times)$ concepts is given as

$$\mathbf{C} ::= \top \mid \bot \mid \mathsf{N_C} \mid \mathbf{C} \sqcap \mathbf{C} \mid \exists \mathsf{N_R}.\mathbf{C} \mid \exists \mathsf{N_R}.\text{Self} \mid \{\mathsf{N_I}\}.$$

Concepts are interpreted as sets: $\top/\bot$ as the whole/empty set, conjunctions $\sqcap$ as a set intersection, and existential restrictions as sets of individuals with some particular role successor. Nominals $\{a\}$ encode singleton sets. Now a $\mathcal{SROEL}(\sqcap, \times)$ axiom can be an assertion $C(a)$ or $R(a, b)$, a general concept inclusion $C \sqsubseteq D$, or a role inclusion of one of the forms $R \sqsubseteq T$, $R \circ S \sqsubseteq T$, $R \sqcap S \sqsubseteq T$, $C \times D \sqsubseteq T$, $R \sqsubseteq C \times D$ where $C, D \in \mathbf{C}$, $R, S, T \in \mathsf{N_R}$, $a, b \in \mathsf{N_I}$.

$\mathcal{SROEL}(\sqcap, \times)$ ontologies are sets of $\mathcal{SROEL}(\sqcap, \times)$ axioms that satisfy some additional properties regarding simplicity or roles and admissibility of range restrictions. Entailment of $\mathcal{SROEL}(\sqcap, \times)$ is defined model theoretically, as usual.

DLP and OWL 2 RL

Description Logic Programs (DLPs) were proposed as the intersection of Description Logics and Horn rules [Gro+03]. The syntax is defined by distinguishing concepts in the head (C_h) and concepts in the body (C_b): Consider a vocabulary ($\mathsf{N_I}$, $\mathsf{N_C}$, $\mathsf{N_R}$). Then, for A and P being an atomic concept and atomic role, respectively, we define roles R, head concepts C_h and body concepts C_b:

$$\begin{aligned} R &::= P \mid P^- \\ C_h &::= C_h \sqcap C_h \mid \forall R.C_h \mid A \mid \bot \mid \top \\ C_b &::= C_b \sqcap C_b \mid C_b \sqcup C_b \mid \exists R.C_b \mid A \mid \bot \mid \top \end{aligned}$$

Then TBox axioms in DLP are of the form $C_b \sqsubseteq C_h$ or $R_1 \circ \cdots \circ R_n \sqsubseteq R$.

DLP is the logic basis of OWL 2 RL that is aimed at applications that require scalable reasoning without sacrificing too much expressive power [Mot+08]. OWL 2 RL is designed to accommodate OWL 2 applications that can trade the full expressivity of the language for efficiency, as well as RDF(S) applications that need some added expressivity. Furthermore, OWL 2 RL reasoning systems can be implemented using rule-based reasoning engines.

2.3.3 Practical Considerations

Based on the strong theoretical foundation of description logics, the Semantic Web community puts tremendous efforts on realizing the automatic reasoning, resulting in

(1) RDF(S), OWL, SPARQL, and RIF specifications as W3C recommendations,

(2) programming tools and application programming interfaces (APIs) for manipulating DL based ontologies and interface to reasoning engines,

(3) and many reasoning engines for various fragments of description logics and OWL.

(1) W3C Recommendations

RDF The Resource Description Framework (RDF) is a standard model for data interchange on the Web [MM04]. An RDF knowledge base is a collection of triples of the form (s, p, o), which means that s and o is related by p. When p is the special role `rdf:type`, the triples $(s, \texttt{rdf:type}, o)$ means that s is an instance of o.

RDFS The RDF Schema (RDFS) is a semantic extension of RDF [BG04]. More built-in properties (e.g., `rdfs:range`, `rdfs:domain`, `rdfs:subClassOf`, and `rdfs:subPropertyOf`) are defined in RDFS. These properties correspond to the DL axioms as shown in Table 2.5.

OWL The OWL Web Ontology Language (latest version OWL 2), is an ontology language for the Semantic Web with formally defined meaning [Gro12]. OWL 2 ontologies provide classes, properties, individuals, and data values.

In practice, a concrete syntax is needed in order to store OWL 2 ontologies and to exchange them among tools and applications. The primary exchange syntax for OWL 2 is RDF/XML, which is compatible with XML serializations of RDF documents, and is indeed the only syntax that must be supported by all OWL 2 tools. There are also other concrete syntaxes that may also be used. These include alternative RDF serializations, such as Turtle; OWL/XML as an XML serialization; and a more "readable" syntax, called the Manchester syntax used in several ontology editing tools. Finally, the functional-style syntax can also be used for serialization, although its main purpose is specifying the structure of the language.

There are two alternative semantics for OWL: the direct semantics [MPG12] (informally OWL 2 DL) and the RDF-based semantics [Sch12] (informally OWL 2 Full). In this thesis we are only interested in the direct semantics as it is compatible with the model theoretic semantics of the description logic $\mathcal{SROIQ}$. The advantage of this close connection is that the extensive description logic literature and implementation experience can be directly exploited by OWL 2 tools. In contrast, the RDF-based Semantics assigns meaning directly to RDF graphs.

SPARQL SPARQL 1.1 is a set of specifications that provide languages and protocols to query and manipulate RDF graph content on the Web or in an RDF store [Gro13]. SPARQL Entailment Regimes defines the semantics of SPARQL queries under entailment regimes such as RDFS and OWL [GO13].

RIF The Rule Interchange Format (RIF) is a W3C recommendation [KB13], which is an XML language for rules.

RDF(S) triple			DL Axiom
a	`rdf:type`	C	$C(a)$
R	`rdfs:range`	C	$\top \sqsubseteq \forall R.C$
R	`rdfs:domain`	C	$\exists R.\top \sqsubseteq C$
C	`rdfs:subClassOf`	D	$C \sqsubseteq D$
R	`rdfs:subPropertyOf`	S	$R \sqsubseteq S$

Table 2.5: RDF(S) triples and DL Axioms

(2) Programming Tools

The semantic web community developed many programming tools[4]. In particular, there are several APIs for RDF, OWL, and SPARQL; we briefly describe some of them below.

OWL API OWL API[5] is now the de facto Java library for OWL [HB11], which is an API for OWL 2 and an efficient in-memory reference implementation. It also implements parsers and writers for OWL in several formats, e.g., RDF/XML, OWL/XML parser, OWL Functional Syntax, Turtle, KRSS parser, and OBO Flat file format. Finally, it provides a reasoner interface that is supported by many DL reasoners such as FaCT++, HermiT, Pellet and Racer.

Jena API Jena[6] is also a widely used API which focuses on RDF(S) and SPARQL [Car+04]. It is an API for reading, processing and writing RDF data in XML, N-triples and Turtle formats and an ontology API for handling OWL and RDFS ontologies. As a query engine, it is compliant with the latest SPARQL 1.1 specification.

Protégé-OWL API Protégé is a popular free, open source ontology editor and knowledge-base framework[7]. The Protégé-OWL API is an open-source Java library for the OWL and RDF(S). The API provides classes and methods to load and save OWL files, to query and manipulate OWL data models, and to perform reasoning. Furthermore, the API is optimized for the implementation of graphical user interfaces. Protégé-OWL API is built on top of OWL API.

(3) OWL Reasoners

Designing efficient algorithms for DLs and implementing them in OWL reasoners are central central tasks in the DL and OWL Community. A comprehensive list[8] of reasoners is maintained by Uli Sattler at the University of Manchester. Moreover, there is a dedicated OWL Reasoner Evaluation Workshop (ORE) for OWL reasoner competition.

[4] http://www.w3.org/2001/sw/wiki/Category:Tool
[5] http://owlapi.sourceforge.net/
[6] http://jena.apache.org/
[7] http://protege.stanford.edu/
[8] http://www.cs.man.ac.uk/~sattler/reasoners.html

We describe some well-known reasoners below, focusing on the expressivity, supported API, underlying algorithm, and we will also give some further remarks.

(3.1) Reasoners for expressive DLs

Pellet Pellet[9] fully supports OWL 2 DL and DL-safe SWRL rules [Sir+07]. It can be used with OWL API, Jena API, proprietary Pellet API, Protégé and command line interface (CLI). The underlying algorithm is tableau-based and Pellet is fully implemented in Java. It is open-source under AGPL Version 3, and also under alternative commercial incenses.

HermiT HermiT is an OWL 2 compliant reasoner[10] and supports DL-safe SWRL rules. It can be used with OWL API, Protégé, and command line interface (CLI). The system is based on a novel "hypertableau" calculus[MSH09]. It is open-source and released under LGPL.

KAON2 KAON2[11] is a reasoner for $\mathcal{SHIQ}(D)$, DL-safe SWRL and function free F-Logic. It can be used with its proprietary KAON2 Java API and CLI. The system reduces a $\mathcal{SHIQ}(D)$ ontology to a disjunctive datalog ($\mathsf{Datalog}^{\vee}$) program, and applies the magic set optimization [HMS04]. It is geared towards efficient ABox reasoning for ontologies with large ABoxes.

RacerPro RacerPro[12] is a commercial reasoner that implements an optimized tableau calculus for $\mathcal{SHIQ}$ [Haa+12]. It offers reasoning services for multiple TBoxes and for multiple ABoxes as well. RacerPro provides another query language (nRQL, new Racer Query Language), which also supports negation as failure, numeric constraints w.r.t. attribute values of different individuals, substring properties between string attributes, etc. It can be used with DIG and CLI.

(3.2) EL Reasoners

The dedicated EL reasoners are usually for classifying large TBox by a polynomial algorithm.

CEL CEL[13] is the first dedicated reasoner for classification of $\mathcal{EL}$ ontologies [MS09]. The main module is implemented in Common Lisp. CEL supports OWL API and can be used with Protege. A Java port of CEL is implemented in the JCEL Project[14].

ELK ELK[15] currently supports a part of the OWL 2 EL ontology language [KKS11]. The primary goal of ELK is high performance in standard reasoning tasks, which is achieved by using efficient consequence-based reasoning algorithms that have been further enhanced to take advantage of modern multi-core processors.

[9] http://clarkparsia.com/pellet/
[10] http://www.hermit-reasoner.com/
[11] http://kaon2.semanticweb.org/
[12] http://www.racer-systems.com/products/racerpro/
[13] https://code.google.com/p/cel/
[14] http://jcel.sourceforge.net/
[15] https://code.google.com/p/elk-reasoner

(3.3) DL-Lite Reasoners

Most of DL-Lite reasoners are aiming for efficient query answering over OWL 2 QL (DL-Lite) ontologies via query rewriting. They accept a TBox and a conjunctive query (usually in SPARQL format or Datalog-like format) and produce the rewritten queries in UCQs or Datalog.

The systems QuOnto[16] [Acc+05], OWLGres [SS08], Nayaya [Dra+09a], Iqaros[17] [VSS12] produce UCQs by (optimized) PerfectRef algorithm; Requiem[18] [PHM09] implements a resolution based algorithm similar to KAON2 and produces UCQs or Datalog depending on the expressivity of the TBox; the systems Presto [RA10] and Prexto [Ros12] produce non-recursive Datalog.

Ontop[19] is a platform to query databases as virtual RDF graphs using SPARQL [RKZ13], which mainly contains two components Quest and ontopPro. Quest is the SPARQL engine supports the OWL 2 QL and RDFS entailment regimes [RC12]; ontopPro is the plugin for Protege.

(3.4) RL Reasoners

RL reasoners are usually used for reasoning in OWL 2 RL and RDF graphs using rules.

Jena Jena API implements a rule-based inference engine for reasoning with RDF and OWL data sources, but may be incomplete for OWL 2 in case of disjunction or negation. It is complete for a practical subset of OWL 2 RL.

BaseVISor BaseVISor[20] is a forward-chaining inference engine specialized to handle facts in the form of RDF triples with support for OWL 2 RL and XML Schema Datatypes.

Oracle Database The native RDFS/OWL inference engine in Oracle Database 11g[21] version supports most of the RL/RDF rules. User-defined rules are supported as well.

2.4 Datalog Family and Logic Programming

Logic programming is a large family of rule-based knowledge representation languages. The research of logic programming starts from the Prolog language [War77; Llo87; Kow88], which is later standardized by ISO [ISO95; ISO00] and implemented in several systems, e.g. SWI-

[16] http://www.dis.uniroma1.it/quonto/
[17] https://code.google.com/p/iqaros/
[18] http://www.cs.ox.ac.uk/isg/tools/Requiem/
[19] http://ontop.inf.unibz.it
[20] http://www.vistology.com/basevisor/basevisor.html
[21] http://www.oracle.com/technetwork/database-options/spatialandgraph/overview/rdfsemantic-graph-1902016.html

Prolog[22], Sicstus Prolog[23], GNU Prolog[24], and XSB[25]. Normally, Prolog programs are evaluated by SLD resolution and backtracking [Llo87], which is sensitive to the order of the atoms in the rules and the order of rules in the programs, and thus not fully declarative. Even worse, for some "bad" programs, e.g. $p \leftarrow not\ p$, the evaluation does not terminate.

To achieve the full declarativity, researchers developed a family of Datalog languages. Syntactically, Datalog programs are essentially the function-free fragments of Prolog programs. The main difference is from the semantics perspective: Datalog and its variants are based on fully declarative model-based semantics, and terminate in general. In this section, we will briefly recall the syntax, semantics and computational complexity of Datalog family, and show how they can be used in practice. Readers may refer to [AHV95; Bar02; EIK09; Fab13] for more extensive introductions.

2.4.1 Syntax

Let $(\mathsf{N_P}, \mathsf{N_C})$ be a function-free first-order vocabulary, consisting of the nonempty finite sets of *predicates* $\mathsf{N_P}$, *constants* $\mathsf{N_C}$, and let $\mathsf{N_V}$ be a set of *variables*. A *term* is either a constant from $\mathsf{N_C}$ or a variable from $\mathsf{N_V}$. An *atom* is defined as $p(t_1, \ldots, t_n)$, where $p \in \mathsf{N_P}$, each $t_1, \ldots, t_n$ is a term, and n is called the arity of p. A *classical literal* is a positive (resp. negative) atom a (resp. $\neg a$). A *negation-as-failure (NAF) literal* is a literal a or a *default-negated literal* $not\ a$. A *rule* r is of the form:

$$a_1 \vee \ldots \vee a_s \ \leftarrow \ b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m, \quad m \geq k \geq 0, s \geq 0, \tag{2.7}$$

where $a_1, \ldots, a_l, b_1, \ldots, b_m$ are literals. We refer to $\{a_1, \ldots, a_s\}$ as the *head* of r, denoted $H(r)$, while the conjunction $b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m$ is the *body* of r. We define $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_k\}$ denoted the *positive* part and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$ denoted the *negative* part of r. We say that

(a) r is a fact if $s = 1$ and $m = 0$. By convention, fact rules can be written as "a.".

(b) r is deterministic (disjunction free) if $s = 1$;

(c) r is positive (negation free) if $k = m$;

(d) r is a Datalog rule if it is both positive and disjunction free.

A Datalog$^{\neg,\vee}$ *program* program P is a finite set of rules of the form 2.7. Furthermore, we say that

(a) P is a Datalog program if all rules in P are Datalog rules;

(b) P is a Datalog$^{\neg}$ program if all rules in P are deterministic;

[22] http://www.swi-prolog.org/
[23] http://sicstus.sics.se
[24] http://www.gprolog.org/
[25] http://xsb.sourceforge.net/

(c) P is a $\mathsf{Datalog}^{\vee}$ program if all rules in P are positive;

Example 2.31. *The following program P_g models the reachability in a graph*

$$reach(X,Y) \leftarrow edge(X,Y).$$
$$reach(X,Y) \leftarrow reach(X,Z), edge(Z,Y).$$

and facts about some edges in a graph:

$$edge(1,2). \quad edge(2,3). \quad edge(3,4). \quad edge(5,6).$$

Among $\mathsf{Datalog}^{\neg}$ programs, stratified programs use restricted negations.

Definition 2.32 (Stratification)**.** *Let S be a set of* $\mathsf{Datalog}^{\neg}$ *rules. A stratification of S is a partition $S_0, \ldots, S_k$ of S such that*

- *For each relation symbol p there is a stratum S_i, such that all clauses of S containing p in their heads are members of S_i. In this case one says that the relation symbol p is defined in stratum S_i.*
- *For each stratum S_j and for each positive literal A in the antecedents of members of S_j, the relation symbol of A is defined in a stratum S_i with $i \leq j$.*
- *For each stratum S_j and for each negative literal $\neg A$ in the antecedents of members of S_j, the relation symbol of A is defined in a stratum S_i with $i < j$.*

A set of $\mathsf{Datalog}^{\neg}$ *rules is called stratified, if there exists a stratification of it; A* $\mathsf{Datalog}^{\neg}$ *program is stratified, if the set of its rules is stratified.*

2.4.2 Semantics

Many semantics for $\mathsf{Datalog}^{\neg,\vee}$ programs are proposed in the literature (see [EIK09] for a survey). For simplicity, we first recall the semantics for the programs with only positive literals (so-called general logic programs in [Llo87]). These semantics can be naturally extended to programs with negative literals [GL91].

Given a program P and a vocabulary $(\mathsf{N_P}, \mathsf{N_C})$, the *Herbrand base* HB_P is the set of all atoms with predicates from $\mathsf{N_P}$ and constant symbols from $\mathsf{N_C}$. When the vocabulary is not explicitly stated, we assume that $\mathsf{N_P}$ and $\mathsf{N_C}$ are all the predicates and constants occuring in the program P.

A term (atom, literal) is *grounded* if there is no variables inside. A ground instance of a rule r is obtained by replacing all the variables with some constants. We denote by $gr(P)$ the set of all ground instances of rules in P (relative to HB_P). A program is ground, if all of its rules are ground; ground Datalog programs are also called *propositional logic programs* .

A *(Herbrand) interpretation* I is a subset of of Herbrand base. An interpretation I is a *model* of a grounded rule $r = H(r) \leftarrow B^+(r), not\ B^-(r)$, if $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$ implies $H(r) \in I$. For a program P, I is a *model* of P, if $I \models r$ for every $r \in P$. For a non-ground P, I is a model of P if it is a model of $gr(P)$.

Example 2.33 (Example 2.31 cont.)**.** *The Herbrand domain of P_g is $HB_{P_g} = \{1, 2, \ldots, 6\}$, and the Herbrand base of P_g is $\{edge(i, j), reach(i, j) \mid i, j \in HB_{P_g}\}$. The ground $gr(P_g)$ of P_g contains the original facts and the following rules*

$$
\begin{aligned}
reach(1,1) &\leftarrow edge(1,1).\\
reach(1,2) &\leftarrow edge(1,2).\\
reach(1,3) &\leftarrow edge(1,3).\\
reach(1,4) &\leftarrow edge(1,4).\\
&\ldots\\
reach(6,6) &\leftarrow edge(6,6).\\
reach(1,1) &\leftarrow reach(1,1), edge(1,1).\\
reach(1,2) &\leftarrow reach(1,2), edge(2,2).\\
reach(1,3) &\leftarrow reach(1,3), edge(3,3).\\
&\ldots\\
reach(6,6) &\leftarrow reach(6,6), edge(6,6).
\end{aligned}
$$

It is easy to check that $I_0 = \{edge(1,2), edge(2,3), edge(3,4), edge(5,6), reach(1,2), reach(2,3), reach(3,4), reach(1,3), reach(2,4), reach(1,4), reach(5,6)\}$ is a model. Furthermore, any interpretation which is a superset of I_0 is also a model of P_g.

Minimal Model Semantics

Definition 2.34. *A model I of a* $\mathsf{Datalog}^{\neg}$ *program P is* minimal*, if P has no model J such that $J \subsetneq I$.*

It is a well-known property that each Datalog program P has some minimal model, which in fact is unique; we denote it with $\mathsf{MM}(P)$. In this case, we write $P \models a$ if $\mathsf{MM}(P) \models a$.

The minimal model of Datalog programs can be computed by a fixpoint operator [Llo87]. Let P be a set of ground Datalog program. Then P defines an operator $T_P : 2^{HB_P} \to 2^{HB_P}$, where 2^{HB_P} denotes the set of all Herbrand interpretations of P, by

$$T_P(I) = \{H(r) \in HB_P \mid H(r) \leftarrow B^+(r) \in P \text{ and } B^+(r) \subseteq I\} \tag{2.8}$$

This operator is called the *immediate consequence operator*. Intuitively, it yields all atoms that can be derived by one step application of rules in P with respect to I. Let T_P^∞ be the limit of the sequence $T_P^0 = \emptyset, T_P^{i+1} = T_P(T_P^i), i \geq 0$. Then the minimal model of a ground positive Datalog program P coincides the fixpoint: $\mathsf{MM}(P) = T_P^\infty$.

Example 2.35. *Consider the program P_g in Example 2.31. The minimal model can be computed by immediate consequence operator on the grounding $gr(P_g)$ as follows:*

$$
\begin{aligned}
T^0_{gr(P_g)} &= \emptyset \\
T^1_{gr(P_g)} &= \{edge(1,2), edge(2,3), edge(3,4), edge(5,6)\} \\
T^2_{gr(P_g)} &= T^1_{gr(P_g)} \cup \{reach(1,2), reach(2,3), reach(3,4), reach(5,6)\} \\
T^3_{gr(P_g)} &= T^2_{gr(P_g)} \cup \{reach(1,3), reach(2,4)\} \\
T^4_{gr(P_g)} &= T^3_{gr(P_g)} \cup \{reach(1,4)\} \\
T^n_{gr(P_g)} &= T^4_{gr(P_g)}, \textit{ for } n \geq 4
\end{aligned}
$$

Then

$$
\begin{aligned}
\mathsf{MM}(P) = T^\infty_P = \{&edge(1,2),\ edge(2,3),\ edge(3,4),\ edge(5,6),\ reach(1,2),\ reach(2,3),\\
&reach(3,4),\ reach(1,3),\ reach(2,4),\ reach(1,4),\ reach(5,6)\}.
\end{aligned}
$$

The minimal model semantics of Datalog programs is widely accepted. However, Datalog$^\neg$ programs may have multiple minimal models and some minimal models are counter-intuitive. For instance, it is easy to check that single rule program $p \leftarrow not\ p$ has a single minimal model $\{p\}$, but p is not supported by the absence of p in the model.

Iterative Fixpoint Semantics for Stratified Datalog$^\neg$ Program

Definition 2.36 ([GL88])**.** *Let I be an interpretation for a* Datalog$^\neg$ *program P. The* GL-reduct P^I *of a program P is the set of* Datalog *rules $H(r) \leftarrow B^+(r)$ such that $r = H(r) \leftarrow B^+(r), not\ B^-(r) \in gr(P)$ and $I \cap B^-(r) = \emptyset$.*

Intuitively, the GL-reduct partially evaluates the negative part of the program P with respect to I by removing (a) rules that cannot be "fired" under the Interpretation I and (b) the negative bodies of the rest of the rules.

Definition 2.37. *Let $S_1, \ldots, S_n$ be a stratification for the program P. Then the semantics of the program P is the set M_n where $M_0 = \emptyset$, and the M_i are defined as follows:*

$$
M_1 = \mathsf{MM}({S_1}^{M_0}), \quad M_2 = \mathsf{MM}({S_2}^{M_1}), \quad \ldots, \quad M_n = \mathsf{MM}({S_{n-1}}^{M_{n-1}}).
$$

We note that the iterative fixpoint semantics for stratified Datalog$^\neg$ Program is well-defined as it does not depend on a concrete stratification [AB88].

Stable Model Semantics The stable model semantics was introduced by Gelfond and Lifschitz [GL88], and later became the theoretical foundation for answer set programming [GL91].

Definition 2.38. *Let P be a* Datalog$^\neg$ *program and $I \subseteq HB_P$ be an interpretation. Then I is a* stable model (or answer set) *of P if $\mathsf{MM}(P^I) = I$.*

A program may have zero, one or many stable models. The set of stable models of a program P is denoted $\mathsf{SM}(P)$. Multiple stable models can be seen as multiple possible worlds. Brave and cautious reasonings are defined naturally: a program P bravely entails a ground term q, denoted $P \models_b q$, if $q \in M$, for some $M \in \mathsf{SM}(P)$; similarly, P cautiously entails q, denoted $P \models_c q$, if $q \in M$, for all $M \in \mathsf{SM}(P)$.

Example 2.39. *Let* $P_1 = \{p \leftarrow not\ q\}$, $P_2 = \{p \leftarrow not\ q.\ \ q \leftarrow not\ p\}$, $P_3 = \{p \leftarrow not\ p\}$. *Then* P_1 *has one stable model* $\{p\}$*;* P_2 *has two stable models* $\{p\}$ *and* $\{q\}$. *Clearly,* $P_2 \models_b p$ *and* $P_2 \models_b q$*, but* $P_2 \not\models_c p$*, and* $P_2 \not\models_c q$. P_3 *has no stable models.*

Well-founded Semantics In contrast to stable model semantics, which may have multiple total models, well-founded semantics produce a single partial model, which can be seen as a three-valued models with truth values True, False and Unknown [GRS91].

The original definition is based on the concept of unfounded set [GRS91]. Here we introduce the alternative definition using the γ operator [BS93].

Definition 2.40. *Let* $\gamma_P(I) = \mathsf{MM}(P^I)$ *and* $\gamma_P^2(I) = \gamma_P(\gamma_P(I))$. *As* γ_P *is anti-monotone,* γ_P^2 *is monotone. The set of* well-founded atoms *of* P*, denoted* $WFS(P)$*, is the least fixed point of* γ_P^2. *We denote with* $P \models^{wf} a$ *that* $a \in WFS(P)$. *The set of* unfounded *atoms of* P*, denoted* $UFS(P)$*, is the complement of greatest fixpoint of* γ_P^2.

Intuitively, the well-founded set are atoms which have be true, the unfounded set are atoms which have to false, and the rest of atoms have truth value unknown. A well-founded model I can be represented as a set of literals $WFM(P) = WFS(P) \cup \{\neg a \mid a \in UFS(P)\}$.

Example 2.41. *Consider the programs in Example 2.39.*

For P_1*, the Herbrand base* $HB_{P_1} = \{p, q\}$. *Then*

- $\gamma_1(\emptyset) = \{p\}$, $\gamma_1^2(\emptyset) = \{p\}$, $lfp(\gamma^2) = \{p\}$. $WFS(P_1) = \{p\}$.
- $\gamma_1(HB_P) = \emptyset$, $\gamma_1^2(\emptyset) = \{p\}$, $gfp(\gamma^2) = \{p\}$. $UFS(P_2) = HB_P \setminus \{p\} = \{q\}$.

So the well-founded model of P_1 *is* $WFM(P_1) = \{p, \neg q\}$.

Similarly, we can show that $WFM(P_2) = \emptyset$*, and* $WFM(I_3) = \emptyset$. *In this case, empty sets means that* p *and* q *have truth values of unknown.*

Semantics of Disjunctions The minimal model semantics and GL-reduction are defined as before for non-disjunctive programs. The stable model semantics can be generalized to $\mathsf{Datalog}^{\neg,\vee}$ programs with disjunctions.

Definition 2.42 ([Prz91])**.** *An interpretation* I *is a (disjunctive) stable model of* P *iff* $M \in \mathsf{MM}(P^I)$*; by* $\mathsf{SM}(P)$ *we denote the set of all stable models of* P.

2.4.3 Computational Complexities

We review computational complexity results of $\mathsf{Datalog}^{\neg}$ as a query language over a plain database (which can be seen as a set of facts), and consider the problems of the ground literal entailment under different semantics. For more details on the complexity results of Datalog and its variants, please refer to [Dan+01; Bry+07]

Formally, the decision problem we study is the following: given a plain database D_{in}, a program P, and a ground literal a, decide whether $D_{in} \cup P \models a$. Following [Var82], there are three main kinds of complexities:

(1) The *data complexity* assumes that the Datalog program P is fixed, and takes varying input databases D_{in} and ground atoms a as inputs.

(2) The *program complexity* assumes that D_{in} are fixed, and takes varying programs P and ground atoms as inputs.

(3) The *combined complexity* takes varying input databases D_{in}, Datalog programs P, and ground atoms a as inputs.

Note that for the problems we consider here, the results of program complexity and combined complexity are identical (under polynomial reductions), since the database D_{in} can always be reduced to some program easily.

Propositional logic programming (i.e. ground Datalog program) is tractable.

Proposition 2.43. *Propositional logic programming is* P*-complete under both data complexity and combined complexity.*

The combined complexity for Datalog is exponentially higher than the data complexity.

Proposition 2.44. Datalog *is* P*-complete under data complexity and* EXPTIME*-complete under combine complexity.*

Well-founded semantics for Datalog$^\neg$ does not increase the complexity compared with minimal models for Datalog.

Proposition 2.45 ([GRS91])**.** *Propositional logic programming with negation under well-founded semantics is* P*-complete. Datalog with negation under well-founded semantics is* P*-complete under data complexity and and* EXPTIME*-complete under combined complexity.*

By the complexity of stable model semantics, we mean the complexity of deciding whether a stable model exists. Datalog$^\neg$ programs under stable model semantics are in general intractable.

Proposition 2.46 ([MT91])**.** *Given a propositional* Datalog$^\neg$ *program P, deciding whether P has a stable model is* NP*-complete.*

For queries under stable models semantics, we need to distinguish cautious reasoning and brave reasoning.

Proposition 2.47 ([MT91])**.** *Cautious (resp. brave) reasoning of propositional logic programming with negation under stable model semantics is co-*NP*-complete (resp.* NP*-complete). Cautions (resp. brave) reasoning of* Datalog$^\neg$ *under stable model semantics is co-*NP*-complete (resp.* NP*-complete) under data complexity and co-*NEXPTIME*-complete (resp.* NEXPTIME*-complete) under combined complexity.*

Proposition 2.48 ([EG95; EGM97]). *Propositional* $\mathsf{Datalog}^{\neg,\vee}$ *program under stable model semantics is* Π_2^p*-complete.* $\mathsf{Datalog}^{\neg,\vee}$ *program under stable model semantics is* Π_2^p*-complete under data complexity and co-*$\mathsf{NEXPTIME}^{\mathsf{NP}}$*-complete under combined complexity.*

2.4.4 Practical Considerations

File Formats

The commonly used format for $\mathsf{Datalog}^{\neg,\vee}$ files inherits the Prolog style. This format is standardized by ASP Standardization Working Group, partially for the ASP Competition[26]; the latest version is ASP-Core-2 Input Language Format[27].

The Rule Interchange Format (RIF) is a W3C recommendation [KB13], which is an XML language for expressing rules which computers can execute. The standard RIF dialects are Core, BLD, and PRD. RIF Core provides "safe" positive datalog with built-ins; RIF BLD (Basic Logic Dialect) is positive Horn logic, with equality and built-ins; PRD (Production Rules Dialect) adds a notion of forward-chaining rules, where a rule fires and then performs some action. The RIF dialect RIF-CASPD[28] is a language for exchanging rules among systems that are based on the ASP paradigm.

Systems

There are many reasoners developed for $\mathsf{Datalog}^{\neg}$. See participants of the ASP Competitions 2011[29] and 2013[30] for an incomplete list.

A $\mathsf{Datalog}^{\neg,\vee}$ reasoner usually contains two components: (1) the grounder and (2) the solver for grounded programs.

LParse LParse is a grounder for answer set programming[31] [Syr01]. Lparse also implements several other semantics (classical negation, partial stable models) by translating them into normal logic programs.

Smodels The program Smodels is an implementation of the stable model semantics for logic programs[32] [Sim00]. Smodels can be used either as a C++ library that can be called from user programs or as a stand-alone program together with a suitable front-end. The main front-end is LParse.

ASSAT ASSAT[33] (Answer Sets by SAT solvers) is a system for computing answer sets of a logic program by using SAT solvers [LZ04]. The system ASSAT(X), depending on the SAT

[26] https://www.mat.unical.it/aspcomp2013
[27] https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.0.pdf
[28] http://ruleml.org/rif/RIF-CASPD.html
[29] https://www.mat.unical.it/aspcomp2011/Participants
[30] https://www.mat.unical.it/aspcomp2013/Participants
[31] http://www.tcs.hut.fi/Software/smodels/
[32] http://www.tcs.hut.fi/Software/smodels/
[33] http://assat.cs.ust.hk/

solver X used, computes the answer set of a ground logic program P by incrementally add loop formulas.

DLV DLV[34] is a deductive database system, based on disjunctive logic programming, which is of the core of the DLV family [Leo+06]. DLV system implements a smart grounder and implements many optimizations (e.g. magic set optimization for non-ground query answering) and offers front-ends to several advanced KR formalisms. DLV supports both answer set semantics and well-founded semantics and offers model generation as well as query answering. DLV has several extensions. For instance,

- dlvex is an extension providing access to external predicates which are supplied via libraries;
- DLT is an extension providing reusable template predicate definitions;
- DLV^{DB} is a further extension which follows a more database-oriented approach .

Potassco Potassco[35] (the Potsdam Answer Set Solving Collection) is a bundle of tools for answer set programming [Geb+12].

- Glingo is a grounder which outputs grounded program in LParse format.
- Clasp is an answer set solver which combines the high-level modeling capacities of ASP with state-of-the-art techniques from the area of Boolean constraint solving. The primary clasp algorithm relies on conflict-driven nogood learning, a technique that was proved successful for satisfiability checking (SAT).
- Clingo stands for clasp on Gringo and combines both systems in a single software.

2.5 Description Logics vs Logic Programming

Description logics and logic programming are two families of logics, and they are different in many aspects; cf. [Bry+07]. Most of these differences are inherent from the differences of first-order logic and logic programming (Section 2.5.1 to 2.5.5). We list some of the differences in the following.

2.5.1 Unique Name Assumption

In logics with the unique name assumption (UNA), different names always refer to different entities in the world . Formally, under UNA, for any interpretation I and constants a, b, $a \neq b$ implies that $a^I \neq b^I$.

[34] http://www.dlvsystem.com/dlv/
[35] http://potassco.sourceforge.net/

In DLs, we normally don't use unique name assumption. Consider an ontology $(\mathcal{T}, \mathcal{A})$, where $\mathcal{A} = \{hasChild(peter, jack), hasChild(peter, john)\}$, and

$$\mathcal{T} = \{\geq 2 hasChild \sqsubseteq HappyParent\}.$$

One may expect that $\mathcal{T} \cup \mathcal{A} \models HappyParent(peter)$. However, this does not hold, because there is no way to exclude the models $\mathcal{I}$ with $jack^{\mathcal{I}} = john^{\mathcal{I}}$. Indeed, if we want the desired result, we have to explicitly add the assertion $jack \neq john$ to the ABox.

In logic programming , we normally use unique name assumption. The above reasoning can be modeled as a rule

$$q: \quad HappyParent(X) \leftarrow hasChild(X, Y_1), hasChild(X, Y_2), Y_1 \neq Y_2.$$

Then $\mathcal{A} \cup \{q\} \models HappyParent(peter)$ holds.

We remark that in lightweight DLs such as DL-Lite the UNA is often adopted, as equality reasoning is expensive [Cal+07].

2.5.2 Open Domain vs Close Domain

The first-order semantics of DLs use open domain, i.e. in an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ the domain $\Delta^{\mathcal{I}}$ is not fixed and could be any set. In DLs, we can talk about anonymous individuals (i.e. unnamed elements in $\Delta^{\mathcal{I}} \setminus \{d^{\mathcal{I}} \mid d \in \mathsf{N_I}\}$). For example, we can express every person has a father: $Person \sqsubseteq \exists hasFather$. Then, informally in every model of this axiom, every person will have a father, even when the individual of his father is not explicitly mentioned in the ontology.

$\mathsf{Datalog}^{\neg}$ programs normally use Herbrand domain, which is a closed domain, once the vocabulary is fixed. We can not talk about nameless individuals in $\mathsf{Datalog}^{\neg}$.

There are some variants of $\mathsf{Datalog}^{\neg}$, like Open Answer Set Programming (OASP) and $\mathsf{Datalog}^{\pm}$, using open domain assumption. But open domains in $\mathsf{Datalog}^{\neg}$ makes the reasoning much more difficult, or even undecidable. Other syntax or semantics restrictions needs to be applied for the decidability. See Section 3.3.2 and 3.3.3 for more discussion.

2.5.3 Open world vs close world

Under the closed world assumption (CWA), certain answers are admitted as a result of failure to find a proof. More specifically, if no proof of a positive ground literal exists, then the negation of that literal is assumed true [Rei87]. In contrast, the open world assumption (OWA) is that the truth-value of a statement is independent of whether or not it is known. The first-order semantics corresponds to OWA; the minimal model nature of logic programming corresponds to CWA.

When the information can be assumed to be complete, the results by CWA is intuitive. For instance, assume that we have a database of all directed flights between cities in the world and there is no information of direct flights between Vienna and Shanghai in the database. If we ask whether some direct flight between Vienna and Shanghai exists, under CWA, the answer is false; under OWA, the answer is unknown, since the flight information might be incomplete.

2.5.4 Strong negation and default negation

Related to CWA, default negation of a statement says it is not known to be true, or can be consistently assumed to be false. In first-order logic, such default negation is not expressible. To derive negated information in FOL, we must explicitly conclude that the statement is false.

For example, modern laws assume the fact that someone is innocent unless proven guilty. In DL, this can not be modeled. While in logic programming, this can be easily expressed by

$$P_i: \quad innocent(X) \leftarrow person(X), not\ guilty(X). \tag{2.9}$$

2.5.5 Monotonicity

An entailment relation $\models_?$ is called monotonic if $K \models_? \alpha$ always implies $K \cup K' \models_? \alpha$; otherwise, it is called non-monotonic. Description logics inherits the monotonicity property from first-order logic. Logic programming is typically nonmonotonic: adding new information may invalidate previously derived conclusions, which can be easily seen in the following example with the program (2.9):

$$P_i \cup \{person(a)\} \models innocent(a)$$
$$P_i \cup \{person(a),\ guilty(a)\} \not\models innocent(a)$$

2.5.6 Arities

In DLs, the predicates are normally restricted to unary predicates (concept) and binary predicates (roles). In logic programming, we can use predicates of arbitrary arities.

Consider a 4-ary Datalog predicate $Person$ where the arguments are intended as $(Id, Name, Father, Mother)$. To express the same structure in DL, we normally decompose it into several relations: a concept $Person$, a data role $hasName$, and two object roles: $hasFather$, $hasMother$.

Note that some DLs consider high-arity predicates, e.g. $\mathcal{DLR}$ [CGL98], but these do not correspond to OWL directly and are out of the scope of this thesis.

CHAPTER 3

Hybrid Knowledge Bases

As shown in the previous section, description logics and logic programming are two families of KR languages, with different expressivities and properties. It is a natural idea to combine them to realize hybrid knowledge bases in order to use the features from both worlds. However, because of the mismatch of the two families trivial combinations easily lead to undecidability or poor expressivity. Many formalisms of hybrid KBs were proposed by researchers; c.f. surveys [Ant+05; Dra+09b; de +09]. We will briefly survey the state of the art of the formalisms of hybrid KBs in this chapter and in particular we will focus on the formalism of dl-programs.

Informally, a hybrid knowledge base is a pair $\mathcal{KB} = (L, P)$, where L is a DL ontology and P is a logic program. The approaches of hybrid knowledge bases fall into three categories, following the representational paradigms of the respective approaches [de +09]:

- The *loose coupling approaches* (e.g., dl-programs [Eit+08a], F-Logic# KB [Hey+10], and CQ-Programs [Eit+08b]) define the interface between the two formalisms based on *the exchange of the entailment*.
- The *tight coupling approaches* (e.g. SWRL [Hor+04], ELP [KRH08a], $\mathcal{DL}$+*log* [Ros06]) define the interface based on *common models*.
- The *embedding approaches* (e.g. Hybrid MKNF [MR10], FO(ID) [VDB10]) define the interface based on embeddings of both the ontology and the rules in *a single unifying non-monotonic formalism*.

3.1 Loose Coupling Approaches

The loose coupling approaches define the interface between the two formalisms based on the exchange of the entailment. The semantics of the ontology and the rule part are modular and clearly separated.

3.1.1 DL-Programs

The approach of dl-programs [Eit+08a; Eit+11] supports a loosely-coupled integration of rules and ontologies, and provide an expressive combination framework based on the interaction of rules with a DL ontology via so-called *dl-atoms*. Such dl-atoms query the DL KB by checking for entailment of ground atoms or axioms w.r.t. the KB; as knowledge deduced by the rules can be streamed up to the DL KB in turn, a bi-directional flow of information is possible.

Syntax

The formalism of dl-programs have rules similar as logic programs with negation as failure , but the rule bodies may also contain *queries to the DL ontology* in their bodies.

Suppose $(\mathsf{N_P}, \mathsf{N_C})$ is a vocabulary of finite predicate symbol set $\mathsf{N_P}$ and constant symbol set $\mathsf{N_C}$, and let $\mathsf{N_V}$ be a set of variables. As usual, elements from $\mathsf{N_C} \cup \mathsf{N_V}$ are *terms*, and classical atoms have the form $p(t_1, \ldots, t_n)$, where $p \in \mathsf{N_P}$ has arity n and all t_i are terms.

Queries to L occur in so-called dl-atoms. A *dl-query* $Q(\mathbf{t})$ is either

(a) a concept inclusion axiom F or its negation $\neg F$; or

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

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

(d) of the forms $= (t_1, t_2)$ or $\neq (t_1, t_2)$, where t_1 and t_2 are terms.

Note here that $\mathbf{t}$ is the empty argument list in (a), $\mathbf{t} = t$ in (b), $\mathbf{t} = (t_1, t_2)$ in (c) and (d), and terms are defined as above.

Definition 3.1. *A dl-atom has the form*

$$\mathrm{DL}[S_1 op_1 p_1, \ldots, S_m op_m\, p_m; Q](\mathbf{t})\,, \qquad m \geq 0, \tag{3.1}$$

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, if S_i is a concept (a role); and $Q(\mathbf{t})$ is a dl-query. Intuitively, $op_i = \uplus$ (resp. $\cup$) increases S_i (resp. $\neg S_i$) by the extension of p_i and $\cap$ increases S_i by the absence in the extension of p_i.

A dl-rule r *is of the form*

$$a_1 \vee \ldots \vee a_n \;\leftarrow\; b_1, \ldots, b_k, not\, b_{k+1}, \ldots, not\, b_m, \quad m \geq k \geq 0,\ n \geq 0 \tag{3.2}$$

where a_i's are classical atoms and each b_i is either an atom or a dl-atom. As

A dl-program $\mathcal{KB} = (L, P)$ consists of a DL knowledge base L and a finite set of dl-rules P.

Example 3.2. *Let $\mathcal{KB} = (L, P)$ where $L = \{C \sqsubseteq D\}$ and P is the set of rules*

$$p(a). \quad p(b). \quad q(c).$$
$$s(X) \leftarrow \mathrm{DL}[C \uplus p; D](X), \;\; not\; \mathrm{DL}[C \uplus q, C \cup p; D](X)\;\; .$$

Intuitively, we extend in the first dl-atom concept C by predicate p and retrieve then all instances from D in this extended ABox. With the second dl-atom we extend C and $\neg C$ by the extensions of q and p, resp.

The following network example taken from [Dra+09a] is more involved.

Example 3.3. *Suppose that an existing network must be extended by new nodes. The knowledge base Σ contains information about existing nodes $(n_1, \ldots, n_5)$ and their interconnections as well as a definition of "overloaded" nodes (concept $HighTrafficNode$), which are nodes with more than three connections:*

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

The following program P evaluates possible combinations of connecting the new nodes:

$$newnode(x_1). \tag{3.3}$$
$$newnode(x_2). \tag{3.4}$$
$$overloaded(X) \leftarrow \mathrm{DL}[wired \uplus connect; HighTrafficNode](X). \tag{3.5}$$
$$connect(X, Y) \leftarrow newnode(X), \mathrm{DL}[Node](Y), \textit{ not } overloaded(Y), \textit{ not } excl(X, Y). \tag{3.6}$$
$$excl(X, Y) \leftarrow connect(X, Z), \mathrm{DL}[Node](Y), Y \neq Z. \tag{3.7}$$
$$excl(X, Y) \leftarrow connect(Z, Y), newnode(Z), newnode(X), Z \neq X. \tag{3.8}$$
$$excl(x_1, n_4). \tag{3.9}$$

The facts (3.3)-(3.4) (bodyless rules) define the new nodes to be added. Rule (3.5) imports knowledge about overloaded nodes in the existing network, taking new connections already into account. Rule (3.6) 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 (3.7) (there must not be more than one connection for each new node) and Rule (3.8) (two new nodes cannot be connected to the same existing one). Rule (3.9) states a specific condition: node x_1 must not be connected with n_4.

Semantics

The meaning of dl-programs is given by formal semantics, among which (strong and weak) answer set semantics [Eit+08a] and well-founded semantics [Eit+11] are widely used (see [Wan+12] for a survey).

Definition 3.4. *The* Herbrand base *of P, denoted HB_P, is the set of all atoms $p(c_1 \ldots, c_n)$ where $p \in \mathsf{N_P}$ occurs in P and all c_i are from $\mathsf{N_C}$. An* interpretation I relative to P *is any subset of HB_P. Such an I* satisfies (models) *a ground (i.e.,variable-free) atom or dl-atom a under L, denoted $I \models_L a$, if the following holds:*

- *$a \in I$, if $a \in HB_P$;*

- $L(I;\lambda) \models Q(\mathbf{c})$, *where* $\lambda = S_1 op_1\ p_1, \ldots, S_m op_m p_m$, $L(I;\lambda) = L \cup \bigcup_{i=1}^{m} A_i(I)$ *and, for* $1 \leq i \leq m$,

$$A_i(I) = \begin{cases} \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \text{if } op_i = \uplus, \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \text{if } op_i = \cup, \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \notin I\}, & \text{if } op_i = \cap. \end{cases}$$

  if a *is a ground and positive dl-atom* $DL[\lambda; Q](\mathbf{c})$.

We say that the interpretation I *satisfies a ground and disjunction-free dl-rule* r *iff* $I \models_L l$ *for all* $l \in B^+(r)$ *and* $I \not\models_L l$ *for all* $l \in B^-(r)$ *implies* $I \models_L H(r)$. *The interpretation* I *satisfies a* dl*-program* $\mathcal{KB} = (L, P)$, *denoted* $I \models \mathcal{KB}$, *iff* $I \models_L r$ *for every rule* $r \in gr(P)$, *where* $gr(P)$ *is the set of all ground instances of rules in* P *(relative to* HB_P*). We call* $\mathcal{KB}$ satisfiable*, if it has some model, and* unsatisfiable *otherwise.*

A ground dl-atom a is *monotonic* relative to $\mathcal{KB} = (L, P)$ iff $I \subseteq I' \subseteq HB_P$ implies that if $I \models_L a$ then $I' \models_L a$. Otherwise a is nonmonotonic.

Minimal Model Semantics We first lift positive programs to dl-programs. A dl-program $KB = (L, P)$ is positive iff (i) P is "not"-free (ii) every ground dl-atom that occurs in $gr(P)$ is monotonic relative to $\mathcal{KB}$.

It is easy to see that every positive $\mathcal{KB}$ has some model and, like every Datalog program, a *unique minimal (least)* (under inclusion $\subseteq$) model, denoted $\mathsf{MM}(\mathcal{KB})$. This model naturally captures the semantics of positive and stratified dl-programs.

Iterative least model semantics of stratified dl-programs For any dl-program $\mathcal{KB} = (L, P)$, we denote by DL_P the set of all ground dl-atoms that occur in $gr(P)$. We assume that $\mathcal{KB}$ 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 some dl-atom $a \in DL_P$ is a ground literal with an input predicate of a and constant symbols in $\mathsf{N_C}$.

Definition 3.5. *Let* $\mathcal{KB} = (L, P)$ *be a* dl*-program. A stratification of* $\mathcal{KB}$ *(relative to* DL_P^+*) is a mapping* $\mu : HB_P \cup DL_P \to \{0, 1, ..., k\}$ *such that*

(i) for each $r \in gr(P)$, $\mu(H(r)) \geq \mu(l')$ *for each* $l' \in B^+(r)$, *and* $\mu(H(r)) > \mu(l')$ *for each* $l' \in B^-(r)$, *and* $\mu(a) \geq \mu(l)$ *for each input literal* l *of each* $a \in DL_P^+$, *and* $\mu(a) > \mu(l)$ *for each input literal* l *of each* $a \in DL_P^?$.

(ii) We call $k \geq 0$ *the length of* μ. *For every* $i \in \{0, \ldots, k\}$, *we then define the dl-program* $\mathcal{KB}_i$ *as* $(L, P_i) = (L, \{r \in gr(P) \mid \mu(H(r)) = i\})$, *and* HB_{P_i} *(resp.,* $HB^*_{P_i}$ *) as the set of all* $l \in HB_P$ *such that* $\mu(l) = i$ *(resp.,* $\mu(l) \leq i$*).*

We say that a dl-program $\mathcal{KB} = (L, P)$ is stratified, iff it has a stratification μ of some length $k \geq 0$. Its canonical model is determined as follows.

Definition 3.6. *Let* $\mathcal{KB} = (L, P)$ *be a dl-program with a stratification of length* $k \geq 0$. *We define its iterative least models* $M_i \subseteq HB_P$ *with* $i \in \{0, \ldots, k\}$ *by:*

(1) M_0 *is the least model of* $\mathcal{KB}_0$*;*

(2) if $i > 0$*, then* M_i *is the least subset* M *of* HB_P *such that*

(a) M *is a model of* $\mathcal{KB}_i$ *and*

(b) $M \cap HB^*_{P_{i-1}} = M_{i-1} \cap HB^*_{P_{i-1}}$.

Strong Answer Set Semantics The *answer sets* of a general dl-program $\mathcal{KB} = (L, P)$ are defined by a generalized GL reduction to the least model semantics of positive dl-programs.

Definition 3.7. *The* strong dl-transform *of* P *relative to* L *and an interpretation* $I \subseteq HB_P$*, denoted* sP^I_L*, results from* $gr(P)$ *by deleting*

(i) every dl-rule r *such that either* $I \not\models_L a$ *for some* $a \in B^+(r) \cap DL^?_P$ *, or* $I \models_L l$ *for some* $l \in B^-(r)$*; and*

(ii) from each remaining dl-rule r *all literals in* $B^-(r) \cup (B^+(r) \cap DL^?_P)$.

Note that (L, sP^I_L) *has only monotonic dl-atoms and no NAF-literals anymore. Thus,* (L, sP^I_L) *is a positive dl-program and has a unique minimal (the least) model.*

Definition 3.8. *Let* $\mathcal{KB} = (L, P)$ *be a dl-program. A strong answer set of* $\mathcal{KB}$ *is an interpretation* $I \subseteq HB_P$ *such that* I *is the least model of* (L, sP^I_L)*. We write* $\mathcal{KB} \models a$ *for a ground atom* a *if* $I \models_L a$ *for every answer set of* $\mathcal{KB}$*. The brave and cautious reasonings are defined as usual.*

Weak Answer Set Semantics The weak answer set semantics of general dl-programs associates with a dl-program a larger set of models than the strong answer set semantics. It is based on a generalized transformation that removes all NAF-literals and all dl-atoms, and it reduces to the answer set semantics of ordinary programs.

Definition 3.9. *Let* $\mathcal{KB} = (L, P)$ *be a dl-program. The weak dl-transform of* P *relative to* L *and to an interpretation* $I \subseteq HB_P$ *, denoted* wP^I_L *, is the ordinary positive program obtained from* $gr(P)$ *by deleting*

(i) all dl-rules r *such that either* $I \not\models_L a$ *for some dl-atom* $a \notin B^+(r)$ *, or* $I \models_L l$ *for some* $l \in B^-(r)$*; and*

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

Definition 3.10. *Let* $\mathcal{KB} = (L, P)$ *be a dl-program. A weak answer set of* $\mathcal{KB}$ *is an interpretation* $I \subseteq HB_P$ *such that* I *is the least model of the ordinary positive program* wP^I_L*. We denote by* $ans_w(\mathcal{KB})$ *the set of all weak answer sets of* $\mathcal{KB}$*. If a ground literal* l *is in every (resp., some) weak answer set of* $\mathcal{KB}$*, then we say that* l *is a cautious (resp., brave) consequence of* $\mathcal{KB}$ *(under the weak answer set semantics), in symbols* $\mathcal{KB} \models_{w,c} l$ *(resp.,* $\mathcal{KB} \models_{w,b} l$*).*

Example 3.11. *Consider* $P = \{p(a) \leftarrow DL[c \uplus p; c](a)\}$ *and* $L = \emptyset$*. The unique strong answer set of* (L, P) *is* $M_1 = \emptyset$*, while the weak answer sets of* (L, P) *are* M_1 *and* $M_2 = \{p(a)\}$.

Well-founded Semantics The well-founded semantics for dl-programs is defined using the γ^2 operator [Eit+11].

Definition 3.12. *Define the operator $\gamma_{\mathcal{KB}}$ on interpretations I of $\mathcal{KB}$ by $\gamma_{\mathcal{KB}}(I) = \mathsf{MM}(\mathcal{KB}^I)$. As $\gamma_{\mathcal{KB}}$ is anti-monotone, $\gamma^2_{\mathcal{KB}}(I) = \gamma_{\mathcal{KB}}(\gamma_{\mathcal{KB}}(I))$ is monotone and has a least fixpoint, which is the set of* well-founded *atoms of $\mathcal{KB}$, denoted $WFS(\mathcal{KB})$ [Eit+11]; we denote with $\mathcal{KB} \models_{wf} a$ that $a \in WFS(\mathcal{KB})$. The set of unfounded atoms of $\mathcal{KB}$, denoted $UFS(\mathcal{KB})$, is the complement of greatest fixpoint of $\gamma^2_{\mathcal{KB}}$. Finally, the well-founded model of $\mathcal{KB}$ is defined as $WFM(\mathcal{KB}) = WFS(\mathcal{KB}) \cup \{\neg a \mid a \in UFS(\mathcal{KB}))\}$*

Example 3.13. *The dl-program $\mathcal{KB}$ from Example 3.2 has the single answer set $\{p(a), p(b), q(c), s(a), s(b)\}$, which coincides with $WFS(\mathcal{KB})$. If we replace the facts for p in P by the "guessing" rules*

$$p(a) \leftarrow \mathit{not}\ p(b); \quad p(b) \leftarrow \mathit{not}\ p(a),$$

the resulting $\mathcal{KB}$ has the two answer sets $\{p(a), q(c), s(a)\}$ and $\{p(b), q(c), s(b)\}$, while $q(c)$ is the only well-founded atom.

Example 3.14. *The dl-program in Example 3.3 has four strong answer sets (we show only atoms with predicate connect) : $M_1 = \{connect(x_1, n_1), connect(x_2, n_4), \ldots\}$, $M_2 = \{connect(x_1, n_1), connect(x_2, n_5), \ldots\}$, $M_3 = \{connect(x_1, n_5), connect(x_2, n_1), \ldots\}$, and $M_4 = \{connect(x_1, n_5), connect(x_2, n_4), \ldots\}$. Note that the ground DL-atom*

$$DL[wired \uplus connect; HighTrafficNode](n_2)$$

from rule (3.5) is true in any partial interpretation of P. According to the well-founded semantics, the atom $overloaded(n_2)$ is thus true in the well-founded model.

Computational Complexities

We summarize the computational complexity results of the dl-programs for answer sets existence, brave and cautious reasoning, and literal entailment under well-founded semantics in Table 3.1 [Eit+08a; Eit+11].

Systems

Several systems implement dl-programs.

NLP-DL NLP-DL [Eit+04a; Eit+08a] is a proof-of-concept system for dl-programs, supporting both well-founded semantics and answer set semantics. This prototype evaluation algorithm uses the DL reasoner RacerPro and the deductive database system DLV and is programmed in PHP scripts with an online demo [1].

[1] https://www.mat.unical.it/ianni/swlp/

dl-program	$\mathcal{SHIF}$	$\mathcal{SHOIN}$
KB positive	EXPTIME-complete	NEXPTIME-complete
KB stratified	EXPTIME-complete	$\mathsf{P}^{\mathsf{NEXPTIME}}$-complete
KB general	NEXPTIME-complete	$\mathsf{P}^{\mathsf{NEXPTIME}}$-complete

(a) Complexity of deciding strong or weak answer set existence

dl-program	$\mathcal{SHIF}$	$\mathcal{SHOIN}$
KB positive	EXPTIME-complete	co-NEXPTIME-complete
KB stratified	EXPTIME-complete	$\mathsf{P}^{\mathsf{NEXPTIME}}$-complete
KB general	co-NEXPTIME-complete	$\mathsf{P}^{\mathsf{NEXPTIME}}$-complete

(b) Complexity of cautious reasoning from the strong or weak answer sets

dl-program	$\mathcal{SHIF}$	$\mathcal{SHOIN}$
KB positive	EXPTIME-complete	$\mathsf{D}^{\mathsf{EXPTIME}}$-complete / $\mathsf{P}^{\mathsf{NEXPTIME}}$-complete
KB stratified	EXPTIME-complete	$\mathsf{P}^{\mathsf{NEXPTIME}}$-complete
KB general	NEXPTIME-complete	$\mathsf{P}^{\mathsf{NEXPTIME}}$-complete

(c) Complexity of brave reasoning from the strong / weak answer sets

dl-program	$\mathcal{SHIF}$	$\mathcal{SHOIN}$
KB general	EXPTIME-complete	$\mathsf{P}^{\mathsf{EXPTIME}}$-complete
Data Complexity	P^{NP}-complete	P^{NP}-complete

(d) Complexity of literal entailment under the well-founded semantics.

Table 3.1: Computational Complexity Results of dl-programs

DLVHEX DLVHEX[2] [Eit+06], as the successor of NLP-DL, is a prototype implementation written in C++ for computing answer sets of so-called HEX-programs – an extension of dl-programs for reasoning with external sources (not necessarily DL knowledge bases) under the answer set semantics. By using the *Description Logic Plugin* [3], which interfaces to OWL ontologies via a DL reasoner (currently RacerPro), DLVHEX can reason from dl-programs under the answer set semantics.

3.1.2 CQ-Programs

CQ-Programs extend DL-Programs by generalizing dl-query to conjunctive queries [Eit+08b].

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

[3] http://www.kr.tuwien.ac.at/research/systems/dlvhex/dlplugin.html

Definition 3.15 (dl-atom in CQ-Programs)**.** *A dl-atom α is of the form* $\mathrm{DL}[\lambda; q](X)$*, 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 X (in this case, α is called a (u)cq-atom), or*
- *$q(X)$ is a dl-query as in definition 3.1 (in this case, α is called an ordinary dl-atom),*

We define satisfaction of atoms with respect to an interpretation. Let I be an interpretation of P. Then

- the satisfaction of an ordinary dl-atom is the same as that defined in dl-programs;
- a ground instance $a(c)$ of a (U)CQ-atom $a(X) = \mathrm{DL}[\lambda; q](X)$, is satisfied by I under L, denoted $I \models_L a(c)$, if $c \in ans(q(X), L \cup \lambda(I))$.

The semantics of cq-programs can be defined similarly as for dl-programs [Eit+08b].

Example 3.16. *Consider the following cq-program [Eit+08b] which is adapted from a scenario in [MSS05].*

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

$$P_1 = \{BadChild(X) \leftarrow DL[parent](X, Z), DL[parent](Y, Z), DL[hates](X, Y)\}$$

Apart from the ABox assertions, 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, $BadChild(Cain)$ can be concluded, but not $BadChild(Romulus)$. The reason is that the common father of $Romulus$ and $Remus$ is not an named individula and thus not in the Herbrand domain. Thus there is no way to instantiate the variable X in P_1 to the individual for their common father.

Consider another program

$$P_2 = \{BadChild(X) \leftarrow DL[parent(X, Z), parent(Y, Z), hates(X, Y)](X, Y)\}$$

where the body of the rule is a CQ $\{parent(X, Z), parent(Y, Z), hates(X, Y)\}$ to L with distinguished variables X and Y. We then obtain the desired result $BadChild(Romulus)$.

The DL-plugin of DLVHEX supports all forms of dl-atoms of cq-programs by rewriting cq-atoms to corresponding external atoms (and additional auxiliary rules) in a HEX-program. Since RacerPro only supports DL-safe (U)CQs (only named individuals are under consideration) , DLVHEX is also limited to this restricted form of (U)CQs.

3.1.3 F-Logic# KBs

Frame Logic (or F-Logic) provides a logical foundation for frame-based and object-oriented languages for data and knowledge representation [KLW95]. In F-Logic, one can express many features in object oriented languages, such as object identity, complex objects, inheritance, polymorphism, and encapsulation.

F-Logic# knowledge bases [Hey+10] are a framework based on the semantics of the dl-programs, which provides a loose coupling approach to integrating F-logic rules and DL ontologies by allowing rules to query the ontology using external atoms.

A prototype implementation of F-Logic# is based on the OntoBroker inference engine[4] from ontoprise[5] [Hey+10]. OntoBroker consists of two reasoners: (i) OntoBroker F-logic, a sophisticated and fast rule engine, and (ii) OntoBroker-OWL, an OWL-DL reasoner (the successor of KAON2 [Mot06]) for $\mathcal{SHIQ}$). The two reasoners use the same API, which simplifies the implementation of the interfacing mechanisms of F-Logic#.

3.1.4 Defeasible Logic Rules on Top of Ontologies

Defeasible logic is a non-monotonic logic proposed to formalize defeasible reasoning, which has a low computational complexity [Ant+01]. Antoniou proposed the integration of description logics with defeasible reasoning, where ontology predicates are allowed to occur only in rule bodies [Ant02]. Compared with dl-programs, there is no information flow back to the ontology [de +09]. A prototype system DR-Prolog [AB07] is available online[6].

3.2 Tight coupling Approaches

The tight coupling approaches define the interface based on common models, which are models for both the ontology part and for the rule part, by restricting the predicates used in the respective parts. Tight coupling approaches can be classified, depending on the expressivity of the rule component, to (1) the fragments of first order logics, with only positive Datalog style rules, and (2) $\mathcal{DL}$+*log* knowledge bases and its variants, featuring default negations and full answer set semantics in the rule component.

syntactically, we start from three mutually disjoint predicate alphabets: an alphabet of concept names $\mathsf{N_C}$, an alphabet of role names $\mathsf{N_R}$, and an alphabet of Datalog predicates $\mathsf{N_D}$. Predicates in $\mathsf{N_C}$ and $\mathsf{N_R}$ are called DL-predicates. In the most general case of tight coupling approaches, the knowledge base is a pair (L, P) where L be a DL ontology and P is a $\mathsf{Datalog}^{\neg,\vee}$ program. The rules r in P are of the following form

$$p_1(\mathbf{X_1}) \vee \ldots \vee p_n(\mathbf{X_n}) \leftarrow r_1(\mathbf{Y_1}), \ldots, r_m(\mathbf{Y_m}), s_1(\mathbf{Z_1}), \ldots, s_k(\mathbf{Z_k}), \\ not\ u_1(\mathbf{W_1}), \ldots, not\ u_h(\mathbf{W_h}) \tag{3.10}$$

[4] http://www.ontoprise.de/en/home/products/ontobroker/

[5] http://www.ontoprise.de

[6] http://www.csd.uoc.gr/~bikakis/DR-Prolog/

where $p_i \in \mathsf{N_C} \cup \mathsf{N_R} \cup \mathsf{N_P}$, $r_i \in \mathsf{N_P}$, $u_i \in \mathsf{N_P}$, $s_i \in \mathsf{N_C} \cup \mathsf{N_R}$.

There are two important safeness conditions for the decidability in this setting. The first one is condition is called DL-safeness[MSS05; Ros05].

Definition 3.17. *A rule r in the form of (3.10) is called DL-safe if each variable in r occurs in one of the positive non-DL-atom in the rule body, i.e., $\bigcup \mathbf{X_i} \cup \bigcup \mathbf{Y_i} \cup \bigcup \mathbf{Z_i} \cup \bigcup \mathbf{W_i} \subseteq \bigcup \mathbf{Y_i}$. A program P is DL-safe if all its rules are DL-safe.*

Sometimes, we say that the rule is interpreted under DL-safeness restrictions. In this case, we do not really put syntax restrictions. Instead, we semantically assume that the variables can only be bounded the named individuals. This is equivalent to

(1) for every rule appending auxiliary atoms $\mathcal{O}(X)$ to the body for all the variables inside it and

(2) adding facts $\mathcal{O}(a)$ for all the named individual a from the ontologies and program.

Note that conjunctive query is not fully captured under DL-safeness conditions[Ros06], because existential variables can not be bounded the unnamed individual in DL-safe rules. Later the DL-safeness condition were relaxed to weakly DL-safeness [Ros06].

Definition 3.18. *A rule r of the form (3.10) is called weakly DL-safe if every head variable of R must appear in at least one of the positive non-DL-atoms, i.e., $\bigcup \mathbf{X_i} \subseteq \bigcup \mathbf{Y_i}$. A program P is weakly DL-safe if all its rules are weakly DL-safe.*

Similarly as the DL-safeness case, when we say that a rule is interpreted under weakly DL-safeness, we semantically assume that the head variables can only be bounded the named individuals. This is equivalent to

(1) for each rule appending auxiliary atoms $\mathcal{O}(x)$ to the body for all head variables $x \in \mathbf{X_i}$ and

(2) adding facts $\mathcal{O}(a)$ for all the named individual a from the ontologies and rules.

Note that weakly DL-safe rules fully capture conjunctive queries.

3.2.1 First-order Combinations

Description Logics and Datalog rules are two orthogonal fragments of first order logics. First order combinations put them into a single formalisms under first order semantics.

CARIN

CARIN [LR98] is one of the early attempts in combining description logic with first-order Horn rules. It combines the two formalisms by allowing the concepts and roles, defined in the DL ontology, to appear as predicates in the *body* of the Horn rules. The reasoning problem for recursive CARIN-$\mathcal{ALCNR}$ knowledge bases is undecidable. Decidability can be achieved by restricting the syntax.

SWRL

The Semantic Web Rule Language (SWRL) as a W3C Submission[7] aims at combining OWL DL ontologies and Horn rules [HP04b]. The predicates in the rules are restricted to the concepts and roles from the DL ontologies. Unfortunately, SWRL in general is too expressive to be decidable [HP04b]. Intuitively, the undecidability comes from that unnamed individuals entailed from the ontology, which in turn will fire some Horn rules under first-order interpretation and cause undecidability.

The decidability of SWRL in practice is not achieved by restriction on the syntax. Instead, the notion *DL-Safe rules* is often applied, which means rules will be applied only to named individuals in the ontology [MSS05]. SWRL is implemented in several reasoners, e.g., KAON2, Pellet, HermiT, under DL-safeness.

DL Rules

As a fragment of SWRL, DL rules allow for a tight integration with DL knowledge bases [KRH08b; MH12]. DL Rules are Datalog rules which can be rewritten to DL axioms. In other words, DL rules provide syntax sugar for some Datalog rules, and they are essentially "the rules inside the ontology". Therefore DL rules don't actually increase the expressivity of DL ontologies.

For example, a rule

$$profOf(x, z) \leftarrow worksAt(x, y), University(y), supervises(x, z), PhDStudent(z)$$

can be equivalently expressed by the following DL axioms using auxiliary roles S_1 and S_2:

$$S_1 \circ supervises \circ S_2 \sqsubseteq profOf,\ \exists worksAt.University \equiv \exists S_1.Self,\ PhDStudent \equiv \exists S_2.Self.$$

ELP

ELP [KRH08b; KRH08a] is a decidable fragment of SWRL that admits reasoning in polynomial time. ELP is based on the tractable DL $\mathcal{EL}^{++}$ and encompasses an extended notion of DL rules [KRH08b]. Furthermore, ELP extends $\mathcal{EL}^{++}$ with a number of features introduced by OWL 2, such as disjoint roles, local reflexively, certain range restrictions, and the universal role. A reasoning algorithm is based on a translation of ELP to Datalog in a tractable fashion. A prototype system ELLY[8] is implemented using the IRIS datalog reasoner.

Nominal Schema

Nominal Schema [Krö+11] is a description-logic style extension of OWL 2 with nominal schemas which can be used like "variable nominal classes" within axioms. This feature allows ontology languages to express arbitrary DL-safe SWRL rules in their native syntax. Adding nominal schemas to OWL 2 does not increase the worst-case reasoning complexity.

[7] http://www.w3.org/Submission/SWRL
[8] http://elly.sourceforge.net/

For example, the rule

$$C(x) \leftarrow hasParent(x, y), hasParent(x, z), married(y, z)$$

can not be expressed in DL rules. In contrast, using nominal schemas, it can be expressed as

$$\exists hasParent.\{z\} \sqcap \exists hasParent.\exists married.\{z\} \sqsubseteq C.$$

3.2.2 $\mathcal{DL}$+*log* and its Variants

Compared with first-order combinations, the framework of $\mathcal{DL}$+*log* and its variants support more expressive rules, featuring default negations and full answer set semantics. The research of this family was from $\mathcal{AL}$-*log* [Ros99], and was later extended to r-hybrid KBs [Ros05] and $\mathcal{DL}$+*log* [Ros06].

R-hybrid Knowledge Bases

Definition 3.19. *Given a description logic $\mathcal{DL}$, an r-hybrid KB is a pair (K, P), where:*

- *K is a $\mathcal{DL}$ ontology;*
- *P is a set of* $\mathsf{Datalog}^{\neg,\vee}$ *rules, where each rule r is of the form (3.10) and r is both* Datalog *safe and DL-safe.*

The semantics of r-hybrid KBs is defined via common models. Informally, an interpretation $\mathcal{I}$ is a model of an r-hybrid KB (K, P), if the projection of $\mathcal{I}$ on DL predicates is a first order model of K, and the projection of $\mathcal{I}$ on Datalog predicates is a answer set of P. See [Ros05] for the formal definition.

$\mathcal{DL}$+*log* Knowledge Bases

$\mathcal{DL}$+*log* KBs further relax r-hybrid KBs by replacing the DL-safeness condition with the weakly DL-safeness [Ros06]. Weakly DL-safeness conditions provide more expressivity. For instance, conjunctive queries, which can not be directly expressed in r-hybrid KBs, are captured by the framework of $\mathcal{DL}$+*log*.

3.3 Embedding approaches

The embedding approaches define the interface based on *embeddings* of both the ontology and the rules in a *single unifying non-monotonic formalism*. Normally, they start from a very expressive language and then consider various interesting (decidable) fragments.

3.3.1 Hybrid MKNF

Minimal Knowledge and Negation as Failure (MKNF) is a formalism proposed in [Lif91], which captures many non-monotonic logics using two modal operators K and not. Hybrid MKNF knowledge bases is one embedding proposal for combining ontologies and rules [MR07]. The definition of Hybrid MKNF knowledge bases is parametric with respect to the ontology language, in the sense that non-monotonic rules can extend any decidable ontology language. Both stable model [MR07] and well-founded semantics [KAH08] have been defined for hybrid MKNF.

A query-driven procedure for Hybrid MKNF knowledge bases was proposed in [AKS09]. It is sound with respect to the original stable model-based semantics, and is correct with respect to the well-founded semantics. This procedure is able to answer conjunctive queries, and is parametric on an inference engine for reasoning in the ontology language. The procedure is based on an extension of a tabled rule evaluation to capture reasoning within an ontology by modeling it as an interaction with an external oracle.

3.3.2 Open Answer Set Programming

Open Answer Set Programming (OASP) can be seen as a framework to represent integrated combined knowledge bases of ontologies and rules that are not necessarily DL-safe [Hey06]. The framework makes the open-domain assumption and has a rule-based syntax supporting negation under a stable model semantics.

An algorithm for the satisfiability checking of simple conceptual logic programs (SCLP), which are a fragment of OASP, was proposed in [HFE09]; a prototype for SCLP was implemented using BProlog.

3.3.3 $\mathsf{Datalog}^{\pm}$

$\mathsf{Datalog}^{\pm}$ [Cal+10] is a recently introduced family of Datalog-based languages. It is a new framework for tractable ontology querying, and for a variety of other applications. $\mathsf{Datalog}^{\pm}$ extends plain Datalog by features such as existentially quantified rule heads and, at the same time, restricts the rule syntax so as to achieve decidability and tractability. In particular, $\mathsf{Datalog}^{\pm}$ fully covers DL-Lite.

3.3.4 FO(ID)

It is well-known that inductive definitions, such as that of transitive closure, cannot be expressed in first-order logic. The logic FO(ID) is an extension of first-order logic with inductive definition [VDB10]. FO(ID) offers a strong semantic integration of FO and LP, in which the LP component is used to define concepts, and FO component can be used to assert additional properties of both the defined concepts and concepts for which no definition is provided.

DL(ID) is the fragment of FO(ID) restricting the FO part to a DL. For example, $\mathcal{ALCI}$(ID) is the combination of description logic $\mathcal{ALCI}$ with inductive definitions. It is not surprising that such strong integration is undecidable; it is even not semi-decidable. To obtain the decidability, a decidable fragment called *guarded* $\mathcal{ALCI}$(ID) was introduced.

The IDP-system[9] implements the task of model expansion for FO(ID). A program transforming $\mathcal{ALCI}$(ID) syntax to input for this system is available, but only works for reasoning in a fixed, finite domain.

3.3.5 Quantified Equilibrium Logic

Equilibrium logic, based on the least constructive extension of the logic of "here-and-there", was proposed to characterize stable models and answer sets [Pea96]. Quantified Equilibrium Logic (QEL) can serve as a unified framework which embraces classical logic as well as disjunctive logic programs under the (open) answer set semantics [Bru+07]. However, QEL is in general undecidable.

[9] http://dtai.cs.kuleuven.be/krr/software/idp

CHAPTER 4

Inline Evaluation of DL-Programs

The main goal of this thesis is to develop efficient reasoning methods for hybrid knowledge bases, in particular for dl-programs. In this chapter, we first abstractly define the inline evaluation framework for dl-programs, based on the the concept of Datalog-rewritability. Then we apply the inline evaluation on dl-programs over lightweight description logics $\mathcal{LDL}^+$, $\mathcal{EL}$, and more expressive Horn-$\mathcal{SHIQ}$.

4.1 A Framework for Inline Evaluation

The term *inline evaluation* is borrowed from the community of computer programming languages. For example, assume we have a small function for the maximum of two numbers, in C programming language:

```
// max of integers x and y
int max(int x, int y){ return x > y ? x : y; }
```

and a function for the maximum elements of an array of integers.

```
// max of an integer array of size n
int max_array(int array[], int n) {
  int result = INT_MIN;
  for (int i = 0; i < n; i++) {
    result = max(result, array[i]);
  }
  return result;
}
```

The function `max` is called `n` times in the function `max_array`. Besides the actual computation part, every function call needs stack frame manipulation and the function return. Such overhead of function calling can be a potential bottleneck of the efficiency. Even worse, function invocation disrupts compile-time code optimization such as register allocation, code compaction,

common subexpression elimination, and constant propagation. To avoid this, modern C compilers may directly expand such function calls to the concrete expressions, and the gains may be considerable [CH89].

```
int max_array(int array[], int n){
  int result = INT_MIN;
  for (int i = 0; i < n; i++){
    // ``max'' is inline expanded
    result = result > array[i] ? result : array[i];
  }
  return result;
}
```

The situation in dl-programs is conceptually similar. The state of that art reasoners for dl-programs usually need to call the underlying Datalog$^\neg$ reasoners and DL reasoners many (sometimes even exponentially many) times to evaluate the dl-atoms, and such cost can be very expensive. In the following, we introduce the *inline framework* for dl-programs, which intuitively reduce all the calls to DL reasoner to some fragments of Datalog program, so that we can only use one call to Datalog$^\neg$ reasoner for the reasoning in dl-programs.

Let $\mathcal{KB} = (\Sigma, P)$ be a dl-program and let a be a ground atom from HB_P . We define a class of DLs, so-called Datalog-*rewritable* DLs, such that reasoning w.r.t. dl-programs over such DLs becomes reducible to Datalog$^\neg$. In particular, we show that for such Datalog-rewritable DLs, we can reduce a dl-program $\mathcal{KB} = (\Sigma, P)$ to a Datalog$^\neg$ program $\Psi(\mathcal{KB})$ and there is a close relation between the models of $\mathcal{KB}$ and $\Psi(\mathcal{KB})$.

We abstractly define which DLs we consider Datalog-rewritable.

Definition 4.1. *A Description Logic $\mathcal{DL}$ is* Datalog-rewritable *(for instance query) if there exists a transformation $\Phi_{\mathcal{DL}}$ from $\mathcal{DL}$ KBs to* Datalog *programs such that, for any $\mathcal{DL}$ KB Σ,*

(i) $\Sigma \models Q(\mathbf{o})$ *iff* $\Phi_{\mathcal{DL}}(\Sigma) \models Q(\mathbf{o})$ *for any concept or role name Q from Σ, and individual(s) $\mathbf{o}$ from Σ;*

(ii) $\Phi_{\mathcal{DL}}$ *is* modular*, i.e., for $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ an ABox, $\Phi_{\mathcal{DL}}(\Sigma) = \Phi_{\mathcal{DL}}(\mathcal{T}) \cup \mathcal{A}$;*

In other words, a ground atom a is entailed by the $\mathcal{DL}$ KB Σ iff $a \in \mathsf{MM}(\Phi_{\mathcal{DL}}(\Sigma))$, the unique minimal model of the Datalog program $\Phi_{\mathcal{DL}}(\Sigma)$. Furthermore, we refer to a *polynomial* Datalog-rewritable DL $\mathcal{DL}$, if $\Phi_{\mathcal{DL}}(\Sigma)$ for a $\mathcal{DL}$ KB Σ is computable in polynomial time.

It is easy to see that $\Phi_{\mathcal{DL}}$ is *preserving*, i.e., concept- and role names in Σ are mapped to identically named predicates in $\Phi_{\mathcal{DL}}(\Sigma)$. [1] Moreover, if Σ_1 and Σ_2 are disjoint on concept- and role names, then $\Phi_{\mathcal{DL}}(\Sigma_1)$ and $\Phi_{\mathcal{DL}}(\Sigma_2)$ are as well on the corresponding predicates.

We assume w.l.o.g. that both P and $\Phi_{\mathcal{DL}}(\Sigma)$ are *safe* — each variable appears in a positive normal atom in the body — for $\mathcal{KB} = (\Sigma, P)$.

[1] Note that this is implicit in how we wrote *(i)*.

Definition 4.2. *Let $\Lambda_P \triangleq \{\lambda \mid DL[\lambda; Q]$ occurs in $P\}$, i.e., the input signatures appearing in P. The translation of (Σ, P) to a* Datalog$^\neg$ *program is then built up of the following four components:*

- *A* Datalog *program $\bigcup_{\lambda \in \Lambda_P} \Phi_{\mathcal{DL}}(\Sigma_\lambda)$ where Σ_λ is Σ with all concept and role names subscripted with λ. Intuitively, each input signature of a dl-atom in P will influence Σ differently. As we want to cater for these influences in one program, we have to differentiate between the KBs with different inputs.*
- *A* Datalog *program $\rho(\Lambda_P)$ containing for each $\lambda = S_1 \uplus p_1, \ldots, S_m \uplus p_m \in \Lambda_P$ the rules $S_{i\lambda}(\mathbf{X_i}) \leftarrow p_i(\mathbf{X_i})$, $1 \leq i \leq m$, where the arity of $\mathbf{X_i}$ matches the one of S_i. Intuitively, we add the extension of p_i to the appropriate concept or role.*
- *A set T_P of* Datalog *facts $\top(a)$ and $\top^2(a, b)$ for all a, b in the Herbrand domain of P to ensure their introduction in Σ.*
- *Finally, P^{ord} results from replacing each dl-atom $DL[\lambda; Q](\mathbf{t})$ in P with a new atom $Q_\lambda(\mathbf{t})$.*

The transformation of the dl*-program $\mathcal{KB}$ is then defined as*

$$\Psi(\mathcal{KB}) = \bigcup_{\lambda \in \Lambda_P} \Phi_{\mathcal{DL}}(\Sigma_\lambda) \ \cup \ P^{ord} \ \cup \ \rho(\Lambda_P) \ \cup \ T_P \tag{4.1}$$

Example 4.3. *Let $\mathcal{KB} = (\Sigma, P)$ where $\Sigma = \{\ C \sqsubseteq D\ \}$ and*

$$P = \left\{ \begin{array}{l} p(a). \quad s(a). \quad s(b). \\ q \leftarrow DL[C \uplus s; D](a), not\ DL[C \uplus p; D](b) \end{array} \right\}$$

The input signatures are $\Lambda_P = \{\lambda_1 \triangleq C \uplus s, \lambda_2 \triangleq C \uplus p\}$.

- *The ontology Σ is very simple, and we can define $\Phi(\Sigma) = \{D(X) \leftarrow C(X)\}$. Then $\Phi(\Sigma_{\lambda_1}) = \{D_{\lambda_1}(X) \leftarrow C_{\lambda_1}(X)\}$ $\Phi(\Sigma_{\lambda_2}) = \{D_{\lambda_2}(X) \leftarrow C_{\lambda_2}(X)\}$*
- *DL-atoms are transformed to $\rho(\Lambda_P) = \{\ C_{\lambda_1}(X) \leftarrow s(X). \quad C_{\lambda_2}(X) \leftarrow p(X)\ \}$.*
- *$T_P = \{\top(a). \quad \top(b). \quad \top^2(a, a). \quad \top^2(b, b). \quad \top^2(a, b). \quad \top^2(a, a).\}$*
- *Finally, the component P^{ord} consists of $q \leftarrow D_{\lambda_1}(a), not\ D_{\lambda_2}(b)$ and the original facts.*

Note that $\Psi(\mathcal{KB})$ is a Datalog program, if $\mathcal{KB}$ is negation-free, and a stratified Datalog$^\neg$ program, if $\mathcal{KB}$ is stratified (cf. [Eit+08a]); thus, beneficial for evaluation, acyclic negation is fully preserved.

The following property is easily seen.

Proposition 4.4. *Let $\mathcal{KB}$ be a* dl*-program over a polynomial* Datalog*-rewritable DL. Then, $\Psi(\mathcal{KB})$ is constructible in polynomial time.*

Proof. This immediately follows from the definition of $\Psi(\mathcal{KB})$. □

In order to establish the relation between the models of $\mathcal{KB}$ and $\Psi(\mathcal{KB})$, we use the following intermediate lemmas, similarly as for the proof of Theorem 5.12 in [Eit+04b].

For a dl-program $\mathcal{KB} = (\Sigma, P)$ over a Datalog-rewritable DL and an interpretation I over HB_P , we define an interpretation I^Ψ for $\Psi(\mathcal{KB})$:

$$I^\Psi = I \cup \bigcup_{\lambda \in \Lambda_P} \mathsf{MM}(\Phi(\Sigma_\lambda \cup S(\lambda, I)))$$

where

$$S(\lambda, I) = \{S_\lambda(\mathbf{c}) \mid S \uplus p \in \lambda, p(\mathbf{c}) \in I\}.$$

In other words for an interpretation I of the KB $\mathcal{KB}$, we define an interpretation I^Ψ of $\Psi(\mathcal{KB})$ that corresponds to it, i.e., it contains I and the minimal models of the positive programs consisting of the translation of the KB as well as the facts that follow from the particular extensions of the input predicates w.r.t. I.

We further define some shortcuts: $G(I) \triangleq \gamma_{\mathcal{KB}}(I)$ and $G^\Psi(I) = \gamma_{\Psi(\mathcal{KB})}(I)$.

Lemma 4.5. *Let $\mathcal{KB} = (\Sigma, P)$ be a dl-program over a* Datalog*-rewritable DL, $DL[\lambda; Q](\mathbf{c})$ a ground dl-atom from $gr(P)$, and I an interpretation for $\mathcal{KB}$. Then, $I \models_\Sigma DL[\lambda; Q](\mathbf{c})$ iff $I^\Psi \models Q_\lambda(\mathbf{c})$.*

Proof. We prove both sides simultaneously. All the hints are in the square brackets.

$I \models_\Sigma DL[\lambda; Q](\mathbf{c})$
iff *[Σ_λ is an equivalent rewriting of Σ; take $\lambda' = S_{1\lambda} \uplus p_1, \ldots, S_{m\lambda} \uplus p_m$ for $\lambda = S_1 \uplus p_1, \ldots, S_m \uplus p_m \in \Lambda_P$]*
$I \models_{\Sigma_\lambda} DL[\lambda'; Q_\lambda](\mathbf{c})$
iff *[Def. of $\models_{\Sigma_\lambda}$ and with $A_i(I) = \{S_{i\lambda}(\mathbf{c_i}) \mid p_i(\mathbf{c_i}) \in I\}$]*
$\Sigma_\lambda \cup \bigcup_i A_i(I) \models Q_\lambda(\mathbf{c})$
iff *[Rewriting ABox $\bigcup_i A_i(I)$ as axioms]*
$\Sigma_\lambda \cup \{S_{i\lambda}(\mathbf{c_i}) \mid p_i(\mathbf{c_i}) \in I\} \models Q_\lambda(\mathbf{c})$
iff *[Def. 4.1]*
$Q_\lambda(\mathbf{c}) \in \mathsf{MM}(\Phi(\Sigma_\lambda \cup \{S_{i\lambda}(\mathbf{c_i}) \mid p_i(\mathbf{c_i}) \in I\}))$
iff *[using that the Σ_λ disjoint on the concept- and role names, that Φ is preserving, and that Q_λ is a concept- or role name]*
$Q_\lambda(\mathbf{c}) \in \bigcup_{\lambda \in \Lambda_P} \mathsf{MM}(\Phi(\Sigma_\lambda \cup S(\lambda, I)))$
iff *[Q_λ is concept or role name]*
$Q_\lambda(\mathbf{c}) \in I \cup \bigcup_{\lambda \in \Lambda_P} \mathsf{MM}(\Phi(\Sigma_\lambda \cup S(\lambda, I)))$
iff *[Def. of I^Ψ]*
$Q_\lambda(\mathbf{c}) \in I^\Psi$

□

Lemma 4.6. *Let $\mathcal{KB} = (\Sigma, P)$ be a dl-program over a* Datalog*-rewritable DL, and I an interpretation for $\mathcal{KB}$. Then, $G(I)^\Psi = G^\Psi(I^\Psi)$.*

Proof. This follows from Lemma 4.5 and the observation that sP^I_Σ and $(P^{ord})^{I^\Psi}$ have the same rules where the (positive) dl-atoms $DL[\lambda; Q](\mathbf{c})$ in sP^I_Σ are replaced with $Q_\lambda(\mathbf{c})$ in $(P^{ord})^{I^\Psi}$. □

The following result allows us to reduce reasoning with dl-programs to Datalog$^\neg$ under answer set semantics.

Theorem 4.7. *Let $\mathcal{KB} = (L, P)$ be a dl-program over a* Datalog*-rewritable DL. Then the answer sets of $\mathcal{KB}$ correspond 1-1 to the answer sets of $\Psi(\mathcal{KB})$:*

(i) every answer set of $\mathcal{KB}$ is extendible to an answer set of $\Psi(\mathcal{KB})$; and

(ii) for every answer set J of $\Psi(\mathcal{KB})$, its restriction $I = J|_{HB_P}$ to HB_P is an answer set of $\mathcal{KB}$.

Proof. We show the two directions.

(i) Suppose that I is an answer set of $\mathcal{KB}$. We show that $I^\Psi \supseteq I$ is an answer set of $\Psi(\mathcal{KB})$.

By the definition of answer set of dl-program, we have $I = \mathsf{MM}(\mathcal{KB}^I)$, or equivalently $I = G(I)$. Then $I^\Psi = G(I)^\Psi$. Since $G(I)^\Psi = G^\Psi(I^\Psi)$ holds by Lemma 4.6, we conclude $I^\Psi = G^\Psi(I^\Psi)$, that is, I^Ψ is a answer set of $\Psi(\mathcal{KB})$.

(ii) It is easy to see that for any interpretation K of dl-programs, $K^\Psi|_{HB_P} = K$, as all the elements in $K^\Psi \setminus K$ are in the form of $P_\lambda(\mathbf{c})$.

Since J is an answer set of $\Psi(\mathcal{KB})$, we have $J = G^\Psi(J)$. As $J = I^\Psi$, we imply $I^\Psi = G^\Psi(I^\Psi)$. By lemma 4.6, $G(I)^\Psi = G^\Psi(I^\Psi)$ holds. Then $I^\Psi = G(I)^\Psi$ follows. We restrict both sides to HB_P and conclude $I = I^\Psi|_{HB_P} = G(I)^\Psi|_{HB_P} = G(I)$, that is, I is an answer set of $\mathcal{KB}$.

□

To prove a similar result for well-founded semantics, we first show the following lemma.

Lemma 4.8. *Let $\mathcal{KB} = (\Sigma, P)$ be a dl-program over a* Datalog*-rewritable DL, and let I be an interpretation for $\mathcal{KB}$. Then,* $\mathrm{LFP}(G^2)^\Psi = \mathrm{LFP}((G^\Psi)^2)$.

Proof. The proof technique is similar as the one used in Proposition B.3 in [Eit+04b], using Lemmas 4.5 and 4.6.

Let $I_0 = \emptyset$. One shows first by induction on $k \geq 0$ that for the k-th powers of $G(I_0)$ and $G^\Psi(I_0^\Psi)$, denoted by $G^k(I_0)$ and $(G^\Psi)^k(I_0^\Psi)$, we have

$$G^k(I_0)^\Psi = (G^\Psi)^k(I_0^\Psi). \tag{4.2}$$

The equality obviously holds for $k = 0$. Given (4.2) holds for k, then for $k + 1$, we have

$$G^{k+1}(I_0)^\Psi = (G(G^k(I_0)))^\Psi.$$

Now, let $I = G^k(I_0)$. Then, by Lemma 4.6, we have

$$G(I)^\Psi = G^\Psi(I^\Psi),$$

since by the induction hypothesis, $G^k(I_0)^\Psi = (G^\Psi)^k(I_0^\Psi)$, we get

$$G(G^k(I_0))^\Psi = G^\Psi((G^\Psi)^k(I_0^\Psi)) = (G^\Psi)^{k+1}(I_0^\Psi),$$

which proves (4.2) for each $k \geq 0$. Furthermore, we have that

$$I_0^\Psi \subseteq (G^\Psi)^{2k}(I_0), \text{ for each } k \geq 0\,. \tag{4.3}$$

Observe indeed that $G^\Psi(I_0)$ contains I_0^Ψ, as well as $(G^\Psi)^2(I_0)$, and that $(G^\Psi)^2$ is monotonic. From (4.3) we conclude that $((G^\Psi)^{2k})(I_0^\Psi)$ and $((G^\Psi)^{2k})(I_0)$ converge to the same limit, which is $\mathrm{LFP}((G^\Psi)^2)$. On the other hand, $G^{2k}(I_0)^\Psi$ converges to $\mathrm{LFP}(G^2)^\Psi$. Thus, we get $\mathrm{LFP}(G^2)^\Psi = \mathrm{LFP}((G^\Psi)^2)$.

□

Theorem 4.9. *Let $\mathcal{KB} = (L, P)$ be a dl-program over a* Datalog*-rewritable DL and a be a ground atom from HB_P. Then,*
$\mathcal{KB} \models^{wf} a$ *iff* $\Psi(\mathcal{KB}) \models^{wf} a$.

Proof. We show both directions.
($\Rightarrow$). Assume $\mathcal{KB} \models^{wf} a$. Then, by the definition of the well-founded semantics for dl-programs, $a \in \mathrm{LFP}(\gamma^2_{\mathcal{KB}}) = \mathrm{LFP}(G^2)$. Since for any interpretation $I \subseteq HB_P$, $I \subseteq I^\Psi$, we have that $a \in \mathrm{LFP}(G^2)^\Psi$. By Lemma 4.8, we have that $a \in \mathrm{LFP}((G^\Psi)^2) = \mathrm{LFP}((\gamma_{\Psi(\mathcal{KB})})^2)$, and thus $\Psi(\mathcal{KB}) \models^{wf} a$.
($\Leftarrow$). Assume $\Psi(\mathcal{KB}) \models^{wf} a$. By the definition of well-founded semantics and Lemma 4.8, we have that $a \in \mathrm{LFP}((\gamma_{\Psi(\mathcal{KB})})^2) = \mathrm{LFP}((G^\Psi)^2) = \mathrm{LFP}(G^2)^\Psi$. Since a is a ground atom from HB_P and thus constructed with a predicate from P, we conclude $a \in \mathrm{LFP}(G^2) = \mathrm{LFP}(\gamma^2_{\mathcal{KB}})$, that is, $\mathcal{KB} \models^{wf} a$.

□

Example 4.10. *The dl-program $\mathcal{KB}$ in Example 4.3 is stratified, and has one stable model $I = \{p(a), s(a), s(b), q\}$. The transformed* Datalog$^\neg$ *program $\Psi(\mathcal{KB})$ has only one model $I^\Psi = I \cup T_P \cup \{C_{\lambda_1}(a), C_{\lambda_1}(b), D_{\lambda_1}(a), D_{\lambda_1}(b), C_{\lambda_2}(a), D_{\lambda_2}(a)\}$. The correspondence between I and I^Ψ is as we expected.*

Compared to dl-programs over $\mathcal{SHIF}$ and $\mathcal{SHOIN}$, the computational complexity results of dl-programs over Datalog-rewritable DLs are lower.

Corollary 4.11. *For any dl-program $\mathcal{KB} = (L, P)$ over a DL $\mathcal{DL}$ and ground atom a from HB_P, deciding $\mathcal{KB} \models^{wf} a$ is (i) data complete for* P*, if $\mathcal{DL}$ is* Datalog*-rewritable and (ii) combined complete for* EXPTIME*, if $\mathcal{DL}$ is polynomial* Datalog*-rewritable.*

Similarly, for any dl-program $\mathcal{KB}$ over a DL $\mathcal{DL}$, deciding the existence of the answer set of $\mathcal{KB}$ is (i) data complete for NP*, if $\mathcal{DL}$ is* Datalog*-rewritable and (ii) combined complete for* NEXPTIME*, if $\mathcal{DL}$ is polynomial* Datalog*-rewritable.*

Proof. All hardness results follow that any Datalog$^\neg$ program P amounts to a dl-program $(\emptyset, P)$ [Eit+04b].

The membership results are based on the correctness of the polynomial reduction of dl-programs to Datalog$^\neg$ under both well-founded semantics (Theorem 4.9) and answer set semantics (Theorem 4.7). □

4.2 $\mathcal{LDL}^+$ and OWL 2 RL

We introduce $\mathcal{LDL}^+$ as a particular Datalog-rewritable DL. This DL has no negation (hence the $+$) and distinguishes between expressions on the left- and right-hand side of axioms. $\mathcal{LDL}^+$ offers expressive concept- and role expressions on the left-hand side of axioms (hence the $\mathcal{L}$ in $\mathcal{LDL}^+$), e.g., qualified number restrictions and transitive closure of roles. The Datalog-rewritability of $\mathcal{LDL}^+$ is interesting in itself, showing how to do reasoning in DLs with expressive constructs efficiently via Logic Programming. As a side result, we obtain that reasoning in $\mathcal{LDL}^+$ is tractable, considering both data and combined complexity; more precisely, we show that it is P-complete in both settings. Despite its low complexity, $\mathcal{LDL}^+$ is still expressive enough to represent many constructs useful in ontology applications [BBL08] such as role equivalences and transitive roles. It turns out that $\mathcal{LDL}^+$ is strongly related to OWL 2 RL.

4.2.1 The Description Logic $\mathcal{LDL}^+$

In this section, we introduce the Description Logic $\mathcal{LDL}^+$ and derive some basic model-theoretic properties.

Basic Definitions

We design $\mathcal{LDL}^+$ by syntactic restrictions on the expressions that occur in axioms, distinguishing between occurrence in the "body" α and the "head" β of an axiom $\alpha \sqsubseteq \beta$. We define

- *h-roles* (h for *head*) E, F to be *role names* P, *role inverses* E^-, *role conjunctions* $E \sqcap F$, and *role top* $\top^2$;
- *b-roles* (b for *body*) E, F are the same as h-roles, plus *role disjunctions* $E \sqcup F$, *role sequences* $E \circ F$, *transitive closures* E^+, *role nominals* $\{(o_1, o_2)\}$, where o_1, o_2 are individuals.

Furthermore, let *basic concepts* C, D be concept names A, the top symbol $\top$, and *conjunctions* $C \sqcap D$; then we define

- *h-concepts* C, D are *basic concepts* B, and *value restrictions* $\forall E.B$ where E a **b-role**;
- *b-concepts* C, D are *basic concepts* B, *disjunctions* $C \sqcup D$, *exists restrictions* $\exists E.C$, *atleast restrictions* $\geq nE.C$, and *nominals* $\{o\}$, where E is a b-role, and o is an individual.

Note that all h-roles are also b-roles, but an analog relation does not hold for concepts: $\forall E.C$ is an h-concept but not a b-concept. When immaterial, we will refer to both b-concepts and h-concepts as ($\mathcal{LDL}^+$) concepts; we use an analog convention for roles.

Now an $\mathcal{LDL}^+$ *KB* is a pair $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ of a finite TBox $\mathcal{T}$ and a finite ABox $\mathcal{A}$, where

- $\mathcal{T}$ is a set of *terminological axioms* $B \sqsubseteq H$, where B is a b-concept and H is an h-concept, and *role axioms* $S \sqsubseteq T$, where S is a b-role and T is an h-role, and
- $\mathcal{A}$ is a set of assertions of the form $C(o)$ and $E(o_1, o_2)$ where C is an h-concept and E an h-role.

Example 4.12. *Reconsider the Example 3.3. It is easily to check that the network ontology Σ apart from the inequalities of the nodes (e.g. $n_1 \neq n_2$) amounts to an $\mathcal{LDL}^+$ KB. The inequalities in Σ are implicitly implied in $\mathcal{LDL}^+$, because of the unique name assumption.*

Normal Form. To simplify matters, we restrict to an expressive normal form of $\mathcal{LDL}^+$ knowledge bases Σ as follows:

- An assertion $C(o)$ is equivalent to the axiom $\{o\} \sqsubseteq C$, and and similarly $E(o_1, o_2)$ is equivalent to $\{(o_1, o_2)\} \sqsubseteq E$; hence, we assume that the ABox is empty and identify Σ with its TBox.
- Every axiom $B \sqsubseteq H$ as above can be equivalently rewritten using the following rewriting rules exhaustively such that H is either a concept name A, the $\top$ symbol, or $\forall E.A$, where A is a concept name and E is a b-role.
  - $B \sqsubseteq C \sqcap D$ could be rewritten as $B \sqsubseteq C$ and $B \sqsubseteq D$;
  - $B \sqsubseteq \forall E.C$ could be rewritten as $B \sqsubseteq \forall E.A$ and $A \sqsubseteq C$ for some new concept name A;
- We can similarly remove conjunction from the head T of role axioms $S \sqsubseteq T$, and restrict the h-role T to role names, inverse role names, and $\top^2$.

Proposition 4.13. *Every $\mathcal{LDL}^+$ KB Σ can be transformed into the form described in polynomial (in fact, in linear) time.*

Proof. It immediately follows from the definition of normal form transformation. □

In the sequel, we tacitly deal with such *normalized* $\mathcal{LDL}^+$ KBs.

Immediate Consequence Operator

In the following, we define an immediate consequence operator for $\mathcal{LDL}^+$ that allows us to calculate the ground entailment of atoms. Moreover, we show that ground entailment for $\mathcal{LDL}^+$ is domain independent, and thus can be confined to the constants in the KB.

We first show that b-concepts satisfy a *monotonicity* property. For a given KB Σ and interpretations $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ and $\mathcal{J} = (\Delta, \cdot^{\mathcal{J}})$ over the same domain Δ, we write $\mathcal{I} \subseteq \mathcal{J}$ if $A^{\mathcal{I}} \subseteq A^{\mathcal{J}}$

for concept names A in Σ and $P^{\mathcal{I}} \subseteq P^{\mathcal{J}}$ for role names P in Σ; note that $o^{\mathcal{I}} = o^{\mathcal{J}}$ for any individual o due to the unique names assumption. Then $\mathcal{I} \subset \mathcal{J}$ if $\mathcal{I} \subseteq \mathcal{J}$ but $\mathcal{I} \neq \mathcal{J}$. We say that a model $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ of Σ is *minimal*, if there is no model $\mathcal{J} = (\Delta, \cdot^{\mathcal{J}})$ of Σ such that $\mathcal{J} \subset \mathcal{I}$.

Definition 4.14. *An $\mathcal{LDL}^+$ concept (role) C (E) is* monotonic*, if for each pair of interpretations $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ and $\mathcal{J} = (\Delta, \cdot^{\mathcal{J}})$ of Σ, $\mathcal{I} \subseteq \mathcal{J}$ implies $C^{\mathcal{I}} \subseteq C^{\mathcal{J}}$ ($E^{\mathcal{I}} \subseteq E^{\mathcal{J}}$).*

Proposition 4.15. *All b-concepts and all $\mathcal{LDL}^+$ roles are monotonic.*

Proof. This can be easily inductively proved on the structure of the concepts and roles. □

Note that an h-concept $\forall E.B$ is not monotonic. For example, take two interpretations I_1, I_2, where $\Delta^{I_1} = \Delta^{I_2} = \{1\}, E^{I_1} = \emptyset, B^{I_1} = \emptyset, E^{I_2} = \{(1,1)\}$, and $B^{I_2} = \emptyset$. It is easy to check that $I_1 \subseteq I_2$, but $(\forall E.B)^{I_1} = \{1\} \not\subseteq \emptyset = (\forall E.B)^{I_2}$.

Recall that we can write interpretations $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ as *set-interpretations* $\{A(x) \mid x \in A^{\mathcal{I}}\} \cup \{P(x,y) \mid (x,y) \in P^{\mathcal{I}}\} \cup \{\{o\}(o)\}$ for concept (role) names A (P), and for individuals o. Instead of $x \in C^{\mathcal{I}}$ ($(x,y) \in E^{\mathcal{I}}$), we write $\mathcal{I} \models C(x)$ ($\mathcal{I} \models E(x,y)$) for concepts (roles) C (E). Note that each such $\mathcal{I}$ contains $\top(x)$ for every $x \in \Delta$ as well as $\top^2(x,y)$ for all $x, y \in \Delta$.

One can see that for a fixed Δ, the set $\mathbf{I_\Delta}$ of all set-interpretations over Δ is under the usual subset relation $\subseteq$ a complete lattice as in [Tar55].

Indeed, it is a non-empty set, $\subseteq$ is a partial order on $\mathbf{I_\Delta}$, and for any two sets $\mathcal{I}$ and $\mathcal{J}$ in $\mathbf{I_\Delta}$ there is a least upper bound $\mathcal{I} \cup \mathcal{J}$ and greatest lower bound $\mathcal{I} \cap \mathcal{J}$. Moreover, it is a complete lattice as every subset $\mathbf{S} \subseteq \mathbf{I_\Delta}$ has a least upper bound $\bigcup \mathbf{S}$ and greatest lower bound $\bigcap \mathbf{S}$. In particular, there are elements $0_\Delta \triangleq \bigcap \mathbf{I_\Delta}$ and $1_\Delta \triangleq \bigcup \mathbf{I_\Delta}$. Note that 0_Δ is the set $\{\{o\}(o) \mid o \text{ ind. in } \Sigma\} \cup \{\top(x), \top^2(x,y) \mid x, y \in \Delta\}$.

For an $\mathcal{LDL}^+$ KB Σ and a domain Δ, we then define an *immediate consequence operator* T_Δ on $\mathbf{I}_\Delta$ as follows, where A ranges over the concept names, P over the role names, and x, y over Δ:

$$\begin{aligned} T_\Delta(\mathcal{I}) = \mathcal{I} &\cup \{A(x) \mid B \sqsubseteq A \in \Sigma, \mathcal{I} \models B(x)\} \\ &\cup \{A(x) \mid B \sqsubseteq \forall E.A \in \Sigma, \mathcal{I} \models B(y), \mathcal{I} \models E(y,x)\} \\ &\cup \{P(x,y) \mid S \sqsubseteq P \in \Sigma, \mathcal{I} \models S(x,y)\} \\ &\cup \{P(y,x) \mid S \sqsubseteq P^- \in \Sigma, \mathcal{I} \models S(x,y)\} \ . \end{aligned}$$

For a set-interpretation $\mathcal{I}$ of Σ over Δ, $T_\Delta(\mathcal{I})$ is still a set-interpretation of Σ over Δ, so the operator T_Δ is well-defined over $\mathbf{I}_\Delta$.

As easily seen, T_Δ is *monotone*, i.e., $\mathcal{J} \subseteq \mathcal{I}$ implies $T_\Delta(\mathcal{J}) \subseteq T_\Delta(\mathcal{I})$, and thus has a least fixpoint $\mathrm{LFP}(T_\Delta)$, i.e., a unique minimal $\mathcal{I}$ such that $T_\Delta(\mathcal{I}) = \mathcal{I}$ [Tar55].

Proposition 4.16. *Let Σ be an $\mathcal{LDL}^+$ KB, Δ a domain. Then, T_Δ is increasing and has a least fixpoint, i.e., there is an $\mathcal{I} \in \mathbf{I}_\Delta$ such that $T_\Delta(\mathcal{I}) = \mathcal{I}$ and no $\mathcal{J} \in \mathbf{I}_\Delta$ with $\mathcal{J} \subset \mathcal{I}$ exists such that $T_\Delta(\mathcal{J}) = \mathcal{J}$.*

Proof. If T_Δ is increasing, the fixpoint result follows directly from [Tar55, Theorem 2]. We show that T_Δ is indeed increasing.

Assume $\mathcal{I} \subseteq \mathcal{J}$, set-interpretations of Σ over Δ; we show that $T_\Delta(\mathcal{I}) \subseteq T_\Delta(\mathcal{J})$. Take an $A(x) \in T_\Delta(\mathcal{I})$, then either (1) $A(x) \in \mathcal{I}$, (2) there is a terminological axiom $B \sqsubseteq A$ such that $\mathcal{I} \models B(x)$, or (3) there is a terminological axiom $B \sqsubseteq \forall E.A$ such that $\mathcal{I} \models B(y)$ and $\mathcal{I} \models E(y, x)$ for some $y \in \Delta$. Since $\mathcal{I} \subseteq \mathcal{J}$, (1) leads immediately to $A(x) \in \mathcal{J} \subseteq T_\Delta(\mathcal{J})$. For (2), we have, due to monotonicity of B that $\mathcal{J} \models B(x)$ such that again $A(x) \in T_\Delta(\mathcal{J})$. For (3), we have again due to monotonicity of E and B, that $\mathcal{J} \models B(y)$ and $\mathcal{J} \models E(y, x)$ such that $A(x) \in T_\Delta(\mathcal{J})$.

The case for a $P(x, y) \in T_\Delta(\mathcal{I})$ can be done similarly.

□

This fixpoint corresponds to a model of Σ, which in fact is the single minimal model of Σ over Δ.

Proposition 4.17. *Let Σ be an $\mathcal{LDL}^+$ KB and let Δ be a domain over Σ. Then, $\mathcal{I} = (\Delta, \cdot^\mathcal{I})$ is a minimal model of Σ iff $\mathcal{I}$ corresponds to the set-interpretation* $\mathrm{LFP}(T_\Delta)$.

Proof. We abbreviate in the following T_Δ with T and $\mathrm{LFP}(T)$ with L.
($\Rightarrow$) Assume $\mathcal{I} = (\Delta, \cdot^\mathcal{I})$ is a minimal model of Σ. We show that $\mathcal{I}$ corresponds to the set-interpretation L. We assume $\mathcal{I}$ is written in its set-interpretation notation and prove that $\mathcal{I}$ is indeed the least fixpoint of T.

1. $\mathcal{I}$ is a fixpoint, i.e., $T(\mathcal{I}) = \mathcal{I}$. Clearly, $\mathcal{I} \subseteq T(\mathcal{I})$ by definition of T. Assume $T(\mathcal{I}) \not\subseteq \mathcal{I}$. Then, there is a $A(x)$ or a $R(x, y)$ in $T(\mathcal{I})$ that is not in $\mathcal{I}$. For the case $A(x)$, we have that there is then (1) a $B \sqsubseteq A \in \Sigma$ such that $\mathcal{I} \models B(x)$ or (2) a $B \sqsubseteq \forall E.A \in \Sigma$ such that $\mathcal{I} \models E(y, x)$ and $\mathcal{I} \models B(y)$ for some $y \in \Delta$. For (1), clearly, then $x \in B^\mathcal{I}$ such that (since $\mathcal{I}$ is a model), $x \in A^\mathcal{I}$ and thus $A(x) \in \mathcal{I}$, a contradiction. For (2), $(y, x) \in E^\mathcal{I}$ and $y \in B^\mathcal{I}$ such that, since $\mathcal{I}$ is a model, $y \in (\forall R.A)^\mathcal{I}$ and thus, with $(y, x) \in E^\mathcal{I}$, that $x \in A^\mathcal{I}$, a contradiction. The case $R(x, y)$ can be done similarly. Thus, $T(\mathcal{I}) = \mathcal{I}$.

2. Assume it is not a least fixpoint, then there is a $\mathcal{J}$ such that $\mathcal{J} \subset \mathcal{I}$ and $T(\mathcal{J}) = \mathcal{J}$. We show that $\mathcal{J} = (\Delta, \cdot^\mathcal{J})$ is then a model of Σ, violating the minimality of $\mathcal{I}$.

   Take a terminological axiom $B \sqsubseteq H \in \Sigma$ and $x \in B^\mathcal{J}$. Then $\mathcal{J} \models B(x)$. Assume $H = A$ for a concept name A, then $H(x) \in T(\mathcal{J})(= \mathcal{J})$ by definition of T such that $x \in H^\mathcal{J}$. Assume $H = \forall E.A$ for a concept name A. Then, $x \in (\forall E.A)^\mathcal{J}$. Indeed, if there is a $(x, y) \in E^\mathcal{J}$, then $\mathcal{J} \models E(x, y)$ such that, by definition of T, $A(y) \in T(\mathcal{J}) = \mathcal{J}$ such that $\mathcal{J} \models A(y)$. Thus, $\mathcal{J} \models (\forall E.A)(x)$ and $x \in (\forall E.A)^\mathcal{J}$. Assume $H = \top$, then the axiom is trivially satisfied.

   Role axioms can be treated similarly.

($\Leftarrow$) Assume $\mathcal{I} = L$. That $\mathcal{I}$ is a minimal model can be shown using similar techniques as in the other direction by proving (1) $\mathcal{I}$ is model and (2) $\mathcal{I}$ is a minimal model.

□

We show that for a given domain Δ, the minimal model is unique, i.e., if both $(\Delta, \mathcal{I})$ and $(\Delta, \mathcal{J})$ are minimal models, then $\mathcal{I} = \mathcal{J}$.

Proposition 4.18. *Let Σ be an $\mathcal{LDL}^+$ KB and let Δ be a domain over Σ with minimal models $\mathcal{I}$ and $\mathcal{J}$ over Δ. Then, $\mathcal{I} = \mathcal{J}$.*

Proof. Suppose that otherwise $\mathcal{I}$ and $\mathcal{J}$ are different minimal models. By Proposition 4.17, we have that $\mathcal{I}$ and $\mathcal{J}$ are least fixpoints of T. Using the monotonicity of the immediate consequence operator, we have $T(\mathcal{I} \cap \mathcal{J}) \subseteq T(\mathcal{I}) = \mathcal{I}$ and $T(\mathcal{I} \cap \mathcal{J}) \subseteq T(\mathcal{J}) = \mathcal{J}$. It follows that $T(\mathcal{I} \cap \mathcal{J}) \subseteq \mathcal{I} \cap \mathcal{J}$. By the definition of the immediate consequence operator, $\mathcal{I} \cap \mathcal{J} \supseteq T(\mathcal{I} \cap \mathcal{J})$. Therefore $T(\mathcal{I} \cap \mathcal{J}) = \mathcal{I} \cap \mathcal{J}$. In other words, $\mathcal{I} \cap \mathcal{J}$ is a fixpoint of T. Since $\mathcal{I} \cap \mathcal{J} \subsetneq \mathcal{I}$, we that conclude the fixpoint $\mathcal{I}$ is not least. This is a contradiction. □

Corollary 4.19. *Let Σ be an $\mathcal{LDL}^+$ KB and let Δ be a domain over Σ. Then, there exists a unique minimal model $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$, denoted $\mathsf{MM}(\Delta, \Sigma)$, that equals $\mathrm{LFP}(T_\Delta)$.*

Proof. By Proposition 4.16, T_Δ has a least fixpoint, which is, by Proposition 4.17, equal to the minimal model. The latter is unique by Proposition 4.18. □

Entailment checking of b-concepts can then in each domain be restricted to the unique minimal model for that domain.

Proposition 4.20. *Let Σ be an $\mathcal{LDL}^+$ KB, C a b-concept, and $o \in \Delta_{\mathcal{H}(\Sigma)}$. Then, $\Sigma \models C(o)$ iff for all Δ, $\mathsf{MM}(\Delta, \Sigma) \models C(o)$.*

Proof. The ($\Rightarrow$) follows immediately.

For ($\Leftarrow$), we show that if minimal models entail $C(o)$, then all models do. Take a model $(\Delta, \cdot^{\mathcal{I}_0})$, then either the corresponding set interpretation $\mathcal{I}_0$ is minimal for the domain Δ or not. If it is, we are done, otherwise, there is a model $(\Delta, \cdot^{\mathcal{I}_1})$ of Σ such that $\mathcal{I}_1 \subset \mathcal{I}_0$ for which one repeats the above reasoning, i.e., eventually, we will have a minimal model $\mathcal{I}_n$ such that $\mathcal{I}_n \subset \ldots \mathcal{I}_1 \subset \mathcal{I}_0$ for which $\mathcal{I}_n \models C(o)$. Since C is monotonic, we have that $\mathcal{I}_0 \models C(o)$. □

Note that the proposition does not necessarily hold if C is an h-concept. For example, consider $\Sigma = \{\{a\} \sqsubseteq A\}$ and the h-concept $C = \forall R.A$, where A is a concept name and R is a role name. Clearly, $\Sigma \not\models \forall R.A(a)$. However, when we consider the domain $\Delta_{\mathcal{H}(\Sigma)} = \{a\}$, $\mathsf{MM}(\Delta_{\mathcal{H}(\Sigma)}, \Sigma) = \{A(a)\}$ and $\mathsf{MM}(\Delta_{\mathcal{H}(\Sigma)}, \Sigma) \models \forall R.A(a)$.

Lemma 4.21. *Let $\Delta_0 \subseteq \Delta$ be two domains, $C(E)$ a b-concept (role) expression, o, o_1, o_2 some individuals in Δ_0, $\mathcal{I}$ ($\mathcal{J}$) an interpretation on domain Δ_0 (Δ), and $\mathcal{J} = \mathcal{I} \cup \{\top(x), \top^2(x, y) \mid x, y \in \Delta\}$. Then $\mathcal{I} \models C(o)$ iff $\mathcal{J} \models C(o)$ and $\mathcal{I} \models E(o_1, o_2)$ iff $\mathcal{J} \models E(o_1, o_2)$.*

Proof. We inductively proof this lemma on the number of connectives in C and E.
Base step. Assume that there is no connectives in concepts C or roles E, i.e., $C = A, \{o\}$ or $\top$ and $E = P$.

(1) If $C = A$, then $\mathcal{I} \models A(x)$ iff $A(x) \in \mathcal{I}$ iff $A(x) \in \mathcal{J}$ iff $\mathcal{J} \models A(x)$.

(2) If $C = \top$, then $\mathcal{I}(\mathcal{J}) \models \top(x)$ always hold.

(3) If $C = \{x\}$, then for $x \in \Delta_0$, $\mathcal{I} \models \{x\}(o)$ iff $x = o$ iff $\mathcal{J} \models \{x\}(o)$.

(4) For $E = P$, then $\mathcal{I} \models A(x, y)$ iff $A(x, y) \in \mathcal{I}$ iff $A(x, y) \in \mathcal{J}$ iff $\mathcal{J} \models A(x, y)$.

Inductive step. Assume that the conclusion holds on the concepts or role with number of connectives $\leq k$, then for the C and E with k connectives:

(1) If $C = C_1 \sqcap C_2$, then

$$\begin{aligned}\mathcal{I} \models (C_1 \sqcap C_2)(o) \text{ iff } & \mathcal{I} \models C_1(o) \text{ and } \mathcal{I} \models C_2(o) \\ \text{iff } & \mathcal{J} \models C_1(o) \text{ and } \mathcal{J} \models C_2(o) \\ \text{iff } & \mathcal{J} \models (C_1 \sqcap C_2)(o).\end{aligned}$$

(2) If $C = C_1 \sqcup C_2$, then this case is similar with above "$\sqcap$" case.

(3) If $C = \exists E.C_1$, then

$$\begin{aligned}\mathcal{I} \models (\exists E.C_1)(o) \text{ iff } & \mathcal{I} \models E(o, o') \text{ and } \mathcal{I} \models C_1(o') \text{ for some } o' \in \Delta_0 \\ \text{iff } & \mathcal{J} \models E(o, o') \text{ and } \mathcal{J} \models C_1(o') \text{ for some } o' \in \Delta_0 \\ \text{iff(*) } & \mathcal{J} \models E(o, o') \text{ and } \mathcal{J} \models C_1(o') \text{ for some } o' \in \Delta \\ \text{iff } & \mathcal{J} \models (\exists E.C_1)(o).\end{aligned}$$

Note that the equivalence (*) relies on the property that for any $o' \in \Delta \setminus \Delta_0$ and concept C_0, we have $\mathcal{J} \not\models C_0(o')$. we can inductively prove this property.

(4) $C = \geq nE.C_1$. Similar with above "$\exists$" case.

(5) For roles, the cases can be proved similarly.

□

Importantly, the only relevant interpretation domain is the Herbrand domain $\Delta_{\mathcal{H}(\Sigma)}$ of the KB Σ.

Proposition 4.22. *Let Σ be an $\mathcal{LDL}^+$ KB, C a b-concept, and $o \in \Delta_{\mathcal{H}(\Sigma)}$. Then, $\Sigma \models C(o)$ iff $MM(\Delta_{\mathcal{H}(\Sigma)}, \Sigma) \models C(o)$.*

Proof. Assume $\Delta_{\mathcal{H}(\Sigma)} = \{o_1, \ldots, o_n\}$, and take an arbitrary domain $\Delta = \{o_1, \ldots, o_n, e_1, \ldots, e_m\}$. Denote $\{\top(x), \top^2(x, y) \mid x, y \in \Delta\}$ by $\top_\Delta$. The zero elements for set-interpretations on $\Delta_{\mathcal{H}(\Sigma)}$ and Δ are $0_{\Delta_{\mathcal{H}(\Sigma)}} = \{\{o\}(o) \mid o \in \Delta_{\mathcal{H}(\Sigma)}\} \cup \top_{\Delta_{\mathcal{H}(\Sigma)}}$ and $0_\Delta = \{\{o\}(o) \mid o \in \Delta_{\mathcal{H}(\Sigma)}\} \cup \top_\Delta$.

We abbreviate in the following $T^k_{\Delta_{\mathcal{H}(\Sigma)}}(0_{\Delta_{\mathcal{H}(\Sigma)}})$ with $\mathcal{I}_k$, $T^k_\Delta(0_\Delta)$ with $\mathcal{J}_k$.

We inductively prove that $\mathcal{I}_k \cup \top_\Delta = \mathcal{J}_k$ for all $k \geq 0$.

For $k = 0$, it is easy to verify that $0_\Delta = 0_{\Delta_{\mathcal{H}(\Sigma)}} \cup \top_\Delta$.

For $k + 1$, take an $A(x) \in \mathcal{I}_{k+1}$, then either (1) $A(x) \in \mathcal{I}_k$, (2) there is a terminological axiom $B \sqsubseteq A$ such that $\mathcal{I}_k \models B(x)$, or (3) there is a terminological axiom $B \sqsubseteq \forall E.A$ such that $\mathcal{I}_k \models B(y)$ and $\mathcal{I}_k \models E(y, x)$ for some $y \in \Delta_{\mathcal{H}(\Sigma)}$. (1) leads immediately to $A(x) \in \mathcal{I}_k \subseteq$

$\mathcal{J}_k \subseteq \mathcal{J}_{k+1}$. For (2), due to $\mathcal{I}_k \models B(x)$, by Lemma 4.21, we also have $\mathcal{J}_k \models B(x)$ so that $A(x) \in \mathcal{J}_{k+1}$. For (3), we again have that $\mathcal{J}_k \models B(y)$ and $\mathcal{J}_k \models E(y, x)$ so that $\mathcal{J}_{k+1} \models A(x)$. The case for a $P(x, y) \in T_\Delta(\mathcal{I})$ can be done similarly. So we have $\mathcal{I}_k \cup \top_\Delta \subseteq \mathcal{J}_k$.

Using the observation that for $e_i, e_j \in \{e_1, \ldots, e_m\}$, $\mathcal{J}_k \not\models A(e_i)$ for concept name A, and $\mathcal{J}_k \not\models P(e_i, e_j)$ for role name P, the other direction $\mathcal{I}_k \cup \top_\Delta \supseteq \mathcal{J}_k$ can be proved similarly,.

Now we have proved that $\mathcal{I}_k \cup \top_\Delta = \mathcal{J}_k$ for all $k \geq 0$. So we have that $\mathrm{LFP}(T_{\Delta_{\mathcal{H}(\Sigma)}}) \cup \top_\Delta = \mathrm{LFP}(T_\Delta)$. By Proposition 4.17, we have $MM(\Delta_{\mathcal{H}(\Sigma)}, \Sigma) \cup \top_\Delta = MM(\Delta, \Sigma)$. Now by Lemma 4.21, it follows that for $o \in \Delta_{\mathcal{H}(\Sigma)}$, we have $MM(\Delta_{\mathcal{H}(\Sigma)}, \Sigma) \models C(o)$ iff $\forall \Delta$, $MM(\Delta, \Sigma) \models C(o)$. Finally, by Proposition 4.20, we have $\Sigma \models C(o)$ iff $MM(\Delta_{\mathcal{H}(\Sigma)}, \Sigma) \models C(o)$. □

Note that $MM(\Delta_{\mathcal{H}(\Sigma)}, \Sigma) = \mathrm{LFP}(T_{\Delta_{\mathcal{H}(\Sigma)}})$ is effectively constructable by fixpoint iteration (for a finite KB in finite time).

Proposition 4.22 is at the core of the argument that $\mathcal{ELI}^+$ is a Datalog-rewritable DL, which we show in the next section.

4.2.2 $\mathcal{ELI}^+$ is Datalog-rewritable

To show that a $\mathcal{ELI}^+$ KB Σ is Datalog-rewritable, we construct a suitable Datalog program $\Phi_{\mathcal{ELI}^+}(\Sigma)$ such that $\Sigma \models Q(\mathbf{o})$ iff $\Phi_{\mathcal{ELI}^+}(\Sigma) \models Q(\mathbf{o})$, whenever Q is a concept- or role name appearing in Σ and $\mathbf{o} \subseteq \Delta_{\mathcal{H}(\Sigma)}$.

Define the *closure* of Σ, $clos(\Sigma)$, as the smallest set containing *(i)* all subexpressions that occur in Σ (both roles and concepts) except value restrictions, and *(ii)* for each role name occurring in Σ, its inverse. The closure is in other words the smallest set satisfying the following conditions:

- $\top$ and $\top^2$ are in $clos(\Sigma)$,
- every concept name A, role name R and its inverted role name R^-, nominal $\{o\}$, and role nominal $\{(o_1, o_2)\}$ appearing in Σ is in $clos(\Sigma)$,
- for each $B \sqsubseteq H$ a terminological axiom in Σ with H a concept name or $H = \top$, $B \in clos(\Sigma)$,
- for each $B \sqsubseteq \forall E.A$ a terminological axiom in Σ, $\{B, E\} \subseteq clos(\Sigma)$,
- for each $S \sqsubseteq T$ a role axiom in Σ, $S \in clos(\Sigma)$,
- for every concept expression D in $clos(\Sigma)$, we have
  - if $D = D_1 \sqcap D_2$, then $\{D_1, D_2\} \subseteq clos(\Sigma)$,
  - if $D = D_1 \sqcup D_2$, then $\{D_1, D_2\} \subseteq clos(\Sigma)$,
  - if $D = \exists E.D_1$, then $\{E, D_1\} \subseteq clos(\Sigma)$,
  - if $D = \geq nE.D_1$, then $\{E, D_1\} \subseteq clos(\Sigma)$,
- for every role expression E in $clos(\Sigma)$, we have

– if $E = F^-$, then $F \in clos(\Sigma)$,
– if $E = E_1 \sqcap E_2$, then $\{E_1, E_2\} \subseteq clos(\Sigma)$,
– if $E = E_1 \sqcup E_2$, then $\{E_1, E_2\} \subseteq clos(\Sigma)$,
– if $E = E_1 \circ E_2$, then $\{E_1, E_2\} \subseteq clos(\Sigma)$.
– if $E = E_1^+$, then $E_1 \subseteq clos(\Sigma)$.

Formally, $\Phi_{\mathcal{LDL}^+}(\Sigma)$ is the following program:

- For each axiom $B \sqsubseteq H \in \Sigma$ where H is a concept name, add the rule

$$H(X) \leftarrow B(X) \tag{4.4}$$

- For each axiom $B \sqsubseteq \forall E.A \in \Sigma$ where A is a concept name, add the rule

$$A(Y) \leftarrow B(X), E(X, Y) \tag{4.5}$$

- For each role axiom $S \sqsubseteq T \in \Sigma$, add

$$T(X, Y) \leftarrow S(X, Y) \tag{4.6}$$

(Here $T = P^-$ may be an inverse for a role name P.)

- For each role name P that occurs in Σ, add the rule

$$P(X, Y) \leftarrow P^-(Y, X) \tag{4.7}$$

- For each concept (role) name or (role) nominal Q (Q') in $clos(\Sigma)$, add the rules

$$\begin{array}{rcl} \top(X) & \leftarrow & Q(X) \\ \top(X) & \leftarrow & Q'(X, Y) \\ \top(Y) & \leftarrow & Q'(X, Y) \end{array} \tag{4.8}$$

This ensures that newly introduced constants, e.g., in the context of dl-programs, are also assigned to $\top$ — a relevant property for modularity.

- To deduce the top role, add

$$\top^2(X, Y) \leftarrow \top(X), \top(Y). \tag{4.9}$$

- Next, we distinguish between the types of concepts D in $clos(\Sigma)$:
  – if $D = \{o\}$, add

$$D(o) \leftarrow \tag{4.10}$$

  – if $D = D_1 \sqcap D_2$, add

$$D(X) \leftarrow D_1(X), D_2(X) \tag{4.11}$$

– if $D = D_1 \sqcup D_2$, add

$$\begin{array}{rcl} D(X) & \leftarrow & D_1(X) \\ D(X) & \leftarrow & D_2(X) \end{array} \tag{4.12}$$

– if $D = \exists E.D_1$, add the rule

$$D(X) \leftarrow E(X, Y), D_1(Y) \tag{4.13}$$

– if $D = \geq n\, E.D_1$, add

$$\begin{array}{rcl} D(X) & \leftarrow & E(X, Y_1), D(Y_1), \ldots, E(X, Y_n), D(Y_n), \\ & & Y_1 \neq Y_2, \ldots, Y_i \neq Y_j, \ldots, Y_{n-1} \neq Y_n \end{array} \tag{4.14}$$

(where $1 \leq i < j \leq n$).

- Finally, for each role $E \in clos(\Sigma)$:

– if $E = \{(o_1, o_2)\}$, add

$$E(o_1, o_2) \leftarrow \tag{4.15}$$

– if $E = F^-$, add

$$E(X, Y) \leftarrow F(Y, X) \tag{4.16}$$

– if $E = E_1 \sqcap E_2$, add

$$E(X, Y) \leftarrow E_1(X, Y), E_2(X, Y) \tag{4.17}$$

– if $E = E_1 \sqcup E_2$, add

$$\begin{array}{rcl} E(X, Y) & \leftarrow & E_1(X, Y) \\ E(X, Y) & \leftarrow & E_2(X, Y) \end{array} \tag{4.18}$$

– if $E = E_1 \circ E_2$, add

$$E(X, Y) \leftarrow E_1(X, Z), E_2(Z, Y) \tag{4.19}$$

– if $E = F^+$, add

$$\begin{array}{rcl} E(X, Y) & \leftarrow & F(X, Y) \\ E(X, Y) & \leftarrow & F(X, Z), E(Z, Y) \end{array} \tag{4.20}$$

The following property is immediate.

Proposition 4.23. *Let Σ be an $\mathcal{LDL}^+$ KB . Then, $\Phi_{\mathcal{LDL}^+}(\Sigma)$ is a* Datalog *program whose size is polynomial in the size of Σ . Furthermore, $\Phi_{\mathcal{LDL}^+}$ is modular.*

Proof. The size of the elements in $clos(\Sigma)$ is linear in Σ . and the size of $\Phi_{\mathcal{LDL}^+}\Sigma$ is polynomial in Σ. The only non-trivial case in showing the latter arises by the addition of rule (4.14) inequalities for a number restriction $\geq nQ.E$. We assume, as is not uncommon in DLs (see, e.g., [Tob00]), that the number n is represented in unary notation

$$\underbrace{11\ldots1}_{n}$$

such that the number of introduced inequalities is quadratic in the size of the number restriction.

The modularity is easy to verify. □

The next result is the main result of this section, and shows that $\Phi_{\mathcal{LDL}^+}(\Sigma)$ works as desired.

Proposition 4.24. *For every (normalized) $\mathcal{LDL}^+$ KB Σ, a concept or role name Q, and $\mathbf{o} \subseteq \Delta_{\mathcal{H}(\Sigma)}$, it holds that $\Sigma \models Q(\mathbf{o})$ iff $\Phi_{\mathcal{LDL}^+}(\Sigma) \models Q(\mathbf{o})$.*

Proof. We only prove the case Q is a concept C, as the case Q is a role can be proved similarly. Due to Proposition 4.22, it suffices to show that $\mathsf{MM}(\Delta_{\mathcal{H}(\Sigma)}, \Sigma) \models C(o)$ iff $\Phi_{\mathcal{LDL}^+}(\Sigma) \models C(o)$. Define $MM \triangleq \mathsf{MM}(\Delta_{\mathcal{H}(\Sigma)}, \Sigma)$ and $\Phi(\Sigma) = \Phi_{\mathcal{LDL}^+}(\Sigma)$.
($\Rightarrow$) Assume $MM \models C(o)$. We define an interpretation M for $\Phi(\Sigma)$ as follows:

$$M \triangleq \{D(a) \mid D \in \mathit{clos}(\Sigma), MM \models D(a)\} \cup \{E(a,b) \mid E \in \mathit{clos}(\Sigma), MM \models E(a,b)\}$$

Clearly, $M \models C(o)$, such that it remains to show that M is a minimal model of $gr(\Phi(\Sigma))$.

1. *M is a model of $gr(\Phi(\Sigma))$.* We check satisfiability of the rules in $gr(\Phi(\Sigma))$.

   - Take a rule $H(a) \leftarrow B(a)$ originating from (4.4) such that $B(a) \in M$. Then, by definition of M, $MM \models B(a)$. Since the rule (4.4) was introduced by an axiom $B \sqsubseteq H \in \Sigma$ and MM is a model of Σ, we have that $MM \models H(a)$ and thus $H(a) \in M$.
   - Take a rule $A(b) \leftarrow B(a), E(a,b)$ originating from (4.5) such that $B(a), E(a,b) \in M$. Then, $MM \models B(a)$ and $MM \models E(a,b)$. Since the rule (4.5) originates from an axiom $B \sqsubseteq \forall E.A$, we have that $MM \models (\forall E.A)(a)$, and thus by definition of a value restriction, $MM \models A(b)$ such that $A(b) \in M$.
   - Rules originating from (4.6) can be done similarly.
   - For a rule $P(a,b) \leftarrow P^-(b,a)$ originating from (4.7) with $P^-(b,a) \in M$, we have that $MM \models P^-(b,a)$ such that $MM \models P(a,b)$ and thus $P(a,b) \in M$.
   - Since $MM \models \top(o_1)$ and $MM \models \top^2(o_1,o_2)$, rules (4.8) and (4.9) are satisfied as well.
   - For a rule (4.10), we have that $MM \models \{o\}(o)$ such that $\{o\}(o) \in M$.
   - For a rule $D(a) \leftarrow D_1(a), D_2(a)$ originating from (4.11) with $D_1(a), D_2(a) \in M$, we have $MM \models D_1(a)$ and $MM \models D_2(a)$ such that $MM \models (D_1 \sqcap D_2)(a)$ and thus $(D_1 \sqcap D_2)(a) \in M$ where $D = D_1 \sqcap D_2$.
   - All remaining rules in $gr(\Phi(\Sigma))$ can be verified similarly.

2. *M is a minimal model of $gr(\Phi(\Sigma))$.* Assume it is not, then there is a model $N \subset M$ of $gr(\Phi(\Sigma))$. Define a set-interpretation NN for Σ:

$$\begin{aligned} NN \triangleq \{&A(a) \mid A(a) \in N, A \text{ concept name}\} \\ &\cup \{P(a,b) \mid P(a,b) \in N, P \text{ role name}\} \\ &\cup \{\top(a_1), \top^2(a_1,a_2) \mid a_1, a_2 \in \Delta_{\mathcal{H}(\Sigma)}\} \\ &\cup \{\{o_1\}(o_1), \{(o_1,o_2)\}(o_1,o_2) \mid o_1, o_2 \in \Delta_{\mathcal{H}(\Sigma)}\} \end{aligned}$$

Clearly, NN is a set-interpretation for Σ over $\Delta_{\mathcal{H}(\Sigma)}$ the Herbrand domain of Σ (and of $\Phi(\Sigma)$). Moreover, one can show — using that N is a model of $gr(\Phi(\Sigma))$ and by induction — that for any b-concept expression D' and for any b-role expression E'

$$NN \models D'(a) \Rightarrow D'(a) \in N \tag{4.21}$$

and

$$NN \models E'(a,b) \Rightarrow E'(a,b) \in N \tag{4.22}$$

We show that $NN \subset MM$ and that NN is a model of Σ, contradicting the minimality of MM, such that M is indeed a minimal model of $gr(\Phi(\Sigma))$.

a) $NN \subset MM$. Note that both have the same domain $\Delta_{\mathcal{H}(\Sigma)}$ and thus NN and MM are equal on

$$\{\top(a_1), \top^2(a_1,a_2) \mid a_1, a_2 \in \Delta_{\mathcal{H}(\Sigma)}\} \cup \{\{o_1\}(o_1), \{(o_1,o_2)\}(o_1,o_2) \mid o_1, o_2 \in \Delta_{\mathcal{H}(\Sigma)}\}$$

We show first that $NN \subseteq MM$. Take a $A(a) \in NN$, then $A(a) \in N \subset M$ such that, by definition of M, $MM \models A(a)$, i.e., $A(a) \in MM$ (MM seen as a set-interpretation); and similarly for a $P(a,b) \in NN$.

Since $N \subset M$ there is a $D(a) \in M \setminus N$ or a $E(a,b) \in M \setminus N$. Assume $D(a) \in M \setminus N$. Then, $MM \models D(a)$ and by (4.21) $NN \not\models D(a)$. Since D is an b-concept expression (M only contains b-concept expressions) it is monotonic (Proposition 4.15) and thus $MM \not\subseteq NN$. Together with $NN \subseteq MM$, we have that $NN \subset MM$.

b) NN is a model of Σ. One can prove this using that N is a model of $gr(\Phi(\Sigma))$ together with (4.21) and (4.22). For example, for an axiom $B \sqsubseteq A$ with concept name A and $NN \models B(a)$, we have that $(A(a) \leftarrow B(a)) \in gr(\Phi(\Sigma))$ and, via (4.21) that $B(a) \in N$. Such that, since N is a model of $gr(\Phi(\Sigma))$, $A(a) \in N$. For concept names A, we then have that $A(a) \in NN$ and thus $NN \models A(a)$.

($\Leftarrow$)
Assume $\Phi_{\mathcal{LDL}^+}(\Sigma) \models C(o)$.
Let $MM' = \mathsf{MM}(\Phi_{\mathcal{LDL}^+}(\Sigma))$ be the minimal model $\Phi_{\mathcal{LDL}^+}(\Sigma)$ of Σ .
We can show that MM' is a minimal model of Σ and it is also minimal using the similar technique in the ($\Rightarrow$) direction. It follows that $\Sigma \models C(o)$ by Proposition 4.22.

□

Corollary 4.25. *$\mathcal{LDL}^+$ is (polynomial)* Datalog*-rewritable.*

Proof. Indeed, take $\Phi_{\mathcal{LDL}^+} = \Phi$ as in Proposition 4.24. □

Example 4.26. *Take the network ontology from Example 3.3.*
(1) The following rules are from the closure of the concepts and roles.
For concept $(\geq 1.wired)$, we add the following rule to $\Phi_{\mathcal{LDL}^+}(\Sigma)$:

$$(\geq 1.wired)(X) \leftarrow wired(X, Y).$$

For concept $\forall wired.Node$*, we add the following rule:*

$$Node(Y) \leftarrow wired(X, Y).$$

For role $wired$*, we have*

$$(wired^{-})(X, Y) \leftarrow wired(Y, X).$$

For concept $(\geq 4wired)$*, we have*

$$(\geq 4wired)(X) \leftarrow wired(X, Y_1), wired(X, Y_2), wired(X, Y_3), wired(X, Y_4), \\ Y_1 \neq Y_2, Y_1 \neq Y_3, Y_1 \neq Y_4, Y_2 \neq Y_3, Y_2 \neq Y_4, Y_3 \neq Y_4.$$

(2) Each TBox axiom is replaced by a rule:
For TBox axiom $\geq 1.wired \sqsubseteq Node$*, we add the following:*

$$Node(X) \leftarrow (\geq 1.wired)(X).$$

For TBox axiom $\top \sqsubseteq \forall wired.Node$*, we add the following rule:*

$$Node(Y) \leftarrow wired(X, Y).$$

For TBox axiom $\geq 4wired \sqsubseteq HighTrafficNode$*, we have*

$$HighTrafficNode(X) \leftarrow (\geq 4wired)(X).$$

(3) Finally, the ABox assertions in Σ *(e.g.,* $Node(n_1)$*) are transformed to* Datalog *facts directly. Note that after transformation,* $n_i \neq n_j$*,* $1 \leq i < j \leq 5$*, is dropped because of the Unique Name Assumption (UNA) adopted by* Datalog.

From the complexity of Datalog, we obtain by Datalog-rewritability of $\mathcal{LDL}^+$ immediately that it is tractable under data complexity. Moreover, due to the structure of $\Phi_{\mathcal{LDL}^+}(\Sigma)$, the same holds under combined complexity.

Corollary 4.27. *For every* $\mathcal{LDL}^+$ *KB* Σ*, concept name* A*, and* $o \in \Delta_{\mathcal{H}(\Sigma)}$*, deciding* $\Sigma \models A(o)$ *is in* P*under both data and combined complexity.*

Proof. All rules in $\Phi_{\mathcal{LDL}^+}(\Sigma)$ except (4.14) can be grounded in polynomial time (they use only constantly many variables). The rule (4.14) can be partially grounded for all values of X; whether the body of such a partially grounded rule can be satisfied in a given set of ground atoms is easily decided in polynomial time; hence, we can compute $\mathsf{MM}(\Phi_{\mathcal{LDL}^+}(\Sigma))$ by simple fixpoint iteration in polynomial time. □

OWL 2 RL. The OWL 2 RL Profile extends so-called *Description Logic Programs* [Gro+03]. The latter have a classical model semantics and correspond to the restriction of $\mathcal{LDL}^+$ to conjunction and disjunction of concepts, exists restrictions, and value restrictions. Thus, Description Logic Programs are a strict subset of $\mathcal{LDL}^+$, missing, e.g., nominals, qualified number restrictions, and role constructors.

Proposition 4.28. *Description Logic Programs are a fragment of $\mathcal{LDL}^+$, and thus polynomially* Datalog*-rewritable.*

Transitive Closure vs Transitive Roles

It is a classical result that transitive closure of a binary relation can not be expressed in first order logic; cf. [Bry+07, p.33–p.35]. In practice, in first-order logic (e.g. in OWL 2 RL), the transitive property is often used as an approximation of transitive closure.

There are some subtle differences between transitive closure and transitive roles. We show how it affects the reasoning results by the following example. Let $L_G = \{edge(a, b), edge(b, c), edge(a, d)\}$ be an ontology about a graph. The task is to compute the reachability between the nodes. In RL, one may attempt to use the following axioms introducing an new property $reach$:

$$\mathcal{T}_G = \{edge \sqsubseteq reach,\ \ trans(reach)\}.$$

With $\mathcal{T}_G$, we can correctly compute all the reachability relations:

$$L_G \cup \mathcal{T}_G \models reach(a, b), reach(a, c), reach(b, c), reach(a, d)$$

However, we cannot conclude that two nodes are unreachable by $\mathcal{T}_G$. For example, the fact that c and d are unreachable is not a logical consequence:

$$L_G \cup \mathcal{T}_G \not\models \neg reach(c, d).$$

Intuitively, the problem comes from no minimality restriction on the first-order models. The pair $(c^{\mathcal{I}}, d^{\mathcal{I}})$ can be added to $reach^{\mathcal{I}}$ for any model $\mathcal{I}$ of $L_G \cup \mathcal{T}_G$ freely while not changing the satisfiability. In contrast, using the transitive closure expression in $\mathcal{LDL}^+$, we produce not only the positive results, but also negative information, e.g.,

$$L_G \models \neg edge^+(c, d).$$

Note that the translation of the transitive closure of a role expression E^+ results in the recursive rules (4.20) such that, in contrast with Description Logic Programs, the transformation $\Phi_{\mathcal{LDL}^+}$ is not a first-order rewriting, justifying the term Datalog*-rewritable*. Although DLs with expressive role constructs such as role sequence, role disjunction and transitive closure tend to become undecidable (e.g., $\mathcal{ALC}^+\mathcal{N}(\circ, \sqcup)$ [BS96]), $\mathcal{LDL}^+$ remains decidable. Moreover, it has a Herbrand domain model property (a finite model property where the domain is the Herbrand domain). Indeed, from [BS96] one can see that the undecidability proofs for expressive DLs extensively use functional restrictions on roles, a feature which $\mathcal{LDL}^+$ cannot express.

Concept constructor	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \langle x, y\rangle \in R^{\mathcal{I}}, \text{ for some } y \in C^{\mathcal{I}}\}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
Axiom	**Syntax**	**Semantics**
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$R(a, b)$	$\langle a, b\rangle \in R^{\mathcal{I}}$
concept inclusion (GCI)	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
role inclusion	$R \sqsubseteq T$	$R^{\mathcal{I}} \subseteq T^{\mathcal{I}}$
generalized role inclusion	$R \circ S \sqsubseteq T$	$\{\langle x, z\rangle \mid \langle x, y\rangle \in R^{\mathcal{I}}, \langle y, z\rangle \in R^{\mathcal{I}} \text{ for some } y\} \subseteq T^{\mathcal{I}}$
role conjunction	$S_1 \sqcap S_2 \sqsubseteq T$	$S_1^{\mathcal{I}} \cap S_2^{\mathcal{I}} \subseteq T^{\mathcal{I}}$
concept production	$C \times D \sqsubseteq T$	$C^{\mathcal{I}} \times D^{\mathcal{I}} \subseteq T^{\mathcal{I}}$
	$R \sqsubseteq C \times D$	$R^{\mathcal{I}} \subseteq C^{\mathcal{I}} \times D^{\mathcal{I}}$

Table 4.1: Syntax and semantics of $\mathcal{SROEL}(\sqcap, \times)$

4.3 OWL 2 EL

The description logic $\mathcal{EL}$ is another important family of lightweight DLs. One feature missing in $\mathcal{LDL}^+$ is the existential quantifier occurring in the right hand side of the concept inclusion axioms. For example, in the well known biomedical ontology Galen [2], axioms in the form of $A \equiv B \sqcap \exists R.C$ are heavily used, e.g.,

$$\textit{LeftEar} \equiv \textit{Ear} \sqcap \exists \textit{hasLeftRightSelector.leftSelection},$$

which cannot be expressed in $\mathcal{LDL}^+$.

We consider the DL $\mathcal{SROEL}(\sqcap, \times)$ [Krö10], whose syntax and semantics are given in Table 4.1. A $\mathcal{SROEL}(\sqcap, \times)$ KB is a set of $\mathcal{SROEL}(\sqcap, \times)$ axioms under the restriction of using non-simple roles [Krö10]. It is a superset of OWL 2 EL [Mot+12] disregarding datatypes, and adds concept production, which can be seen as a generalization of domain and role restriction.

Krötzsch shows a Datalog encoding for $\mathcal{SROEL}(\sqcap, \times)$ describing a proof system [Krö10; Krö11]. Every TBox axiom, ABox axiom, concept name, role name, and individual is transformed to a *fact* by an input translation I_{inst} in Table 4.2. A fixed set P_{inst} in Table 4.2 contains the derivation rules, which are independent of the concrete $\mathcal{SROEL}(\sqcap, \times)$ ontology.

In the following, for simplicity, when we say $\mathcal{EL}$, we always mean $\mathcal{SROEL}(\sqcap, \times)$. For an $\mathcal{EL}$ ontology L, define a Datalog transformation by

$$\Phi_{\mathcal{EL}}(L) = P_{inst} \cup \{I_{inst}(\alpha) \mid \alpha \in L\} \cup \{I_{inst}(s) \mid s \in N_I \cup N_C \cup N_R\} \ . \tag{4.23}$$

[2] http://www.opengalen.org

$$
\begin{array}{rlrlrl}
C(a) & \rightsquigarrow isa(a,C) & R(a,b) & \rightsquigarrow triple(a,R,b) & a \in N_I & \rightsquigarrow nom(a) \\
\top \sqsubseteq C & \rightsquigarrow top(C) & A \sqsubseteq \bot & \rightsquigarrow bot(A) & A \in N_C & \rightsquigarrow cls(A) \\
\{a\} \sqsubseteq C & \rightsquigarrow subClass(a,C) & A \sqsubseteq \{c\} & \rightsquigarrow subClass(A,c) & R \in N_R & \rightsquigarrow rol(R) \\
A \sqsubseteq C & \rightsquigarrow subClass(A,C) & A \sqcap B \sqsubseteq C & \rightsquigarrow subConj(A,B,C) & & \\
\exists R.Self \sqsubseteq C & \rightsquigarrow subSelf(R,C) & A \sqsubseteq \exists R.Self & \rightsquigarrow supSelf(A,R) & & \\
\exists R.A \sqsubseteq C & \rightsquigarrow subEx(R,A,C) & A \sqsubseteq \exists R.B & \rightsquigarrow supEx(A,R,B,e^{A \sqsubseteq \exists R.B}) & & \\
R \sqsubseteq T & \rightsquigarrow subRole(R,T) & R \circ S \sqsubseteq T & \rightsquigarrow subRChain(R,S,T) & & \\
R \sqsubseteq C \times D & \rightsquigarrow supProd(R,C,D) & A \times B \sqsubseteq R & \rightsquigarrow subProd(A,B,R) & & \\
R \sqcap S \sqsubseteq T & \rightsquigarrow subRConj(R,S,T) & & & &
\end{array}
$$

Table 4.2: Input translation I_{inst}

Theorem 4.29 ([Krö10])**.** *Given an $\mathcal{EL}$ ontology L, the transformation $\Phi_{\mathcal{EL}}$ is sound and complete w.r.t. instance checking, i.e., (i) $L \models C(a)$ iff $\Phi_{\mathcal{EL}}(L) \models isa(a,C)$, and (ii) $L \models R(a,b)$ iff $\Phi_{\mathcal{EL}}(L) \models triple(a,R,b)$.*

Since the rules in P_{inst} have constant number of variables, evaluation of the Datalog program $\Phi_{\mathcal{EL}}(L)$ can be done in polynomial time. For more discussion about the complexity and how to extend this calculus to concept classification, see [Krö10; Krö11].

Example 4.30. *Let $L_1 = \{A(a),\ A \sqsubseteq \exists R.B,\ B \sqsubseteq C,\ \exists R.C \sqsubseteq D\}$, and suppose we want to decide $L_1 \models D(a)$. The axioms of L_1 and the signatures N_I, N_C, and N_R are transformed to facts in $\Phi_{\mathcal{EL}}(L_1)$:*

$$
\left\{ \begin{array}{l} isa(a,A);\ supEx(A,R,B,e^{A \sqsubseteq \exists R.B});\ subClass(B,C);\ subEx(R,C,D); \\ nom(a);\ cls(A);\ cls(B);\ cls(C);\ cls(D);\ rol(R) \end{array} \right\} .
$$

Then, P^{inst} is added to $\Phi_{\mathcal{EL}}(L_1)$; in particular, also the rules

$$
\begin{array}{rl}
isa(X,X) & \leftarrow nom(X) \\
isa(X,Z) & \leftarrow subClass(Y,Z), isa(X,Y) \\
isa(X_1,Z) & \leftarrow subEx(V,Y,Z), triple(X_1,V,X_2), isa(X_2,Y) \\
triple(X_1,V,X_2) & \leftarrow supEx(Y,V,Z,X_2), isa(X_1,Y) \\
isa(X_2,Z) & \leftarrow supEx(Y,V,Z,X_2), isa(X_1,Y)
\end{array}
$$

From these rules and the above facts, $isa(a,D)$ is derivable, and thus $\Phi_{\mathcal{EL}}(L_1) \models D(a)$.

Note that strictly, $\Phi_{\mathcal{EL}}(L)$ is not a Datalog rewriting as defined in Definition 4.1. The mismatch is that ABox assertions (e.g., $C(a)$) are transformed into reified versions (e.g., $isa(a,C)$); this is easily fixed by using reification rules

$$
\begin{array}{rl}
P^{re} = & \{C(X) \leftarrow isa(X,C);\quad isa(X,C) \leftarrow C(X) \mid C \in N_C\} \cup \\
& \{R(X,Y) \leftarrow triple(X,R,Y);\quad triple(X,R,Y) \leftarrow R(X,Y) \mid R \in N_R\} \ .
\end{array}
$$

$$
\begin{aligned}
isa(X,X) &\leftarrow nom(X)\\
self(X,V) &\leftarrow nom(X), triple(X,V,X)\\
isa(X,Z) &\leftarrow top(Z), isa(X,Z')\\
isa(X,Y) &\leftarrow bot(Z), isa(U,Z), isa(X,Z'), cls(Y)\\
isa(X,Z) &\leftarrow subClass(Y,Z), isa(X,Y)\\
isa(X,Z) &\leftarrow subConj(Y_1,Y_2,Z), isa(X,Y_1), isa(X,Y_2)\\
isa(X,Z) &\leftarrow subEx(V,Y,Z), triple(X,V,X'), isa(X',Y)\\
isa(X,Z) &\leftarrow subEx(V,Y,Z), self(X,V), isa(X,Y)\\
triple(X,V,X') &\leftarrow supEx(Y,V,Z,X'), isa(X,Y)\\
isa(X',Z) &\leftarrow supEx(Y,V,Z,X'), isa(X,Y)\\
isa(X,Z) &\leftarrow subSelf(V,Z), self(X,V)\\
self(X,V) &\leftarrow supSelf(Y,V), isa(X,Y)\\
triple(X,W,X') &\leftarrow subRole(V,W), triple(X,V,X')\\
self(X,W) &\leftarrow subRole(V,W), self(X,V)\\
triple(X,W,X'') &\leftarrow subRChain(U,V,W), triple(X,U,X'), triple(X',V,X'')\\
triple(X,W,X') &\leftarrow subRChain(U,V,W), self(X,U), triple(X,V,X')\\
triple(X,W,X') &\leftarrow subRChain(U,V,W), triple(X,U,X'), self(X',V)\\
triple(X,W,X) &\leftarrow subRChain(U,V,W), self(X,U), self(X,V)\\
triple(X,W,X') &\leftarrow subRConj(V_1,V_2,W), triple(X,V_1,X'), triple(X,V_2,X')\\
triple(X,W,X') &\leftarrow subProd(Y_1,Y_2,W), isa(X',Y_2), isa(X,Y_1)\\
self(X,W) &\leftarrow subProd(Y_1,Y_2,W), isa(X,Y_2), isa(X,Y_1)\\
isa(X,Z_1) &\leftarrow supProd(V,Z_1,Z_2), triple(X,V,X')\\
isa(X,Z_1) &\leftarrow supProd(V,Z_1,Z_2), self(X,V)\\
isa(X',Z_2) &\leftarrow supProd(V,Z_1,Z_2), triple(X,V,X')\\
isa(X,Z_2) &\leftarrow supProd(V,Z_1,Z_2), self(X,V)\\
self(X,W) &\leftarrow subRConj(V_1,V_2,W), self(X,V_1), self(X,V_2)\\
isa(Y,Z) &\leftarrow isa(X,Y), nom(Y), isa(X,Z)\\
isa(X,Z) &\leftarrow isa(X,Y), nom(Y), isa(Y,Z)\\
triple(Z,U,Y) &\leftarrow isa(X,Y), nom(Y), triple(Z,U,X).
\end{aligned}
$$

Table 4.3: Deduction rules P_{inst}

Then $\Phi'_{\mathcal{EL}}(L) = (\Phi_{\mathcal{EL}}(L) \setminus \{I_{inst}(\alpha) \mid \alpha \in \mathcal{A}\}) \cup P^{re} \cup \mathcal{A}$ is a proper Datalog rewriting. However, in the following, for convenience, we will use $\Phi_{\mathcal{EL}}$, instead of $\Phi'_{\mathcal{EL}}$.

4.4 Horn-$\mathcal{SHIQ}$

The Datalog reduction used in the previous sections for $\mathcal{LDL}^+$ and $\mathcal{SROEL}$ has some nice complexity properties. The reduction can be done in polynomial time, and the resulting Datalog program can also be evaluated in polynomial time. Now we move to a more expressive DL Horn-$\mathcal{SHIQ}$, the Horn fragment of the DL $\mathcal{SHIQ}$. The (combined) complexity of Horn-$\mathcal{SHIQ}$ is EXPTIME-complete ($\supsetneq$ P), so we can not expect a worst-case polynomial Datalog rewriting algorithm resulting Datalog program which can be evaluated in polynomial time. Indeed, simply combining the writing results of $\mathcal{LDL}^+$ and $\mathcal{EL}$ is sound but incomplete [KMR10]. Ortiz *el al.*[ORS10] presents a polynomial rewriting of Horn-$\mathcal{SROIQ}$ to Datalog with large number of variables in the rules, which is mainly of theoretical interest for proving complexity results. Later, a more practical algorithm (still EXPTIME in the worse case) for conjunctive query answering over Horn-$\mathcal{SHIQ}$ is presented in [Eit+12b]. This algorithm has three main components: (1) completion of TBox, (2) rules for ABox completion, and (3) rewriting queries w.r.t to the completed TBox. Components (1) and (2) are sufficient for instance queries and will be presented in the following of this section; component (3) will be discussed in Section 5.3.

4.4.1 Syntax and Semantics of Horn-$\mathcal{SHIQ}$

We first recall the syntax of DLs $\mathcal{SHIQ}$ and Horn-$\mathcal{SHIQ}$. The concept and role expressions allowed in $\mathcal{SHIQ}$ can be found in Table 2.3 of Section 2.3.1.

- An expression $C \sqsubseteq D$, where C, D are concepts, is a *general concept inclusion axiom (GCI)*.
- An expression $r \sqsubseteq s$, where r, s are roles, is a *role inclusion (RI)*.
- A *transitivity axiom* is an expression $trans(r)$, where r is a role. A role s is *transitive* in TBox $\mathcal{T}$ if $trans(s) \in \mathcal{T}$ or $trans(s^-) \in \mathcal{T}$. A role s is *simple* in $\mathcal{T}$ if there is no transitive r in $\mathcal{T}$ s.t. $r \sqsubseteq^*_{\mathcal{T}} s$.

A set $\mathcal{T}$ of axioms of GCIs, RIs and transitivity axioms is a $\mathcal{SHIQ}$ TBox if roles in concepts of the form $\geqslant n\, r.C$ and $\leqslant n\, r.C$ are simple.[3]

A TBox $\mathcal{T}$ is a Horn-$\mathcal{SHIQ}$ TBox (in normalized form), if each GCI in $\mathcal{T}$ takes one the following forms:

$$\begin{array}{llll} \text{(F1)} & A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B, & \text{(F3)} & A_1 \sqsubseteq \forall r.B, \\ \text{(F2)} & A_1 \sqsubseteq \exists r.B, & \text{(F4)} & A_1 \sqsubseteq \leqslant 1\, r.B, \end{array}$$

where $A_1, \ldots, A_n, B$ are concept names and r is a role. Axioms (F2) are called *existential*. Without loss of generality, we treat here only Horn-$\mathcal{SHIQ}$ TBoxes in normalized form; our results generalize to full Horn-$\mathcal{SHIQ}$ by means of TBox *normalization*; see e.g. [Kaz09a; KRH07] for a definition and normalization procedures.

[3] In the literature, some researchers put RIs and transitivity axioms into a separate RBox. For simplicity, we do not use a separate RBox in this section.

A Horn-$\mathcal{ALCHIQ}$ TBox is a Horn-$\mathcal{SHIQ}$ TBox with no transitivity axioms. Horn-$\mathcal{ALCHIQ}^{\sqcap}$ TBoxes are obtained by allowing *role conjunction* $r_1 \sqcap r_2$, where r_1, r_2 are roles and in any interpretation $\mathcal{I}$, $(r_1 \sqcap r_2)^{\mathcal{I}} = r_1^{\mathcal{I}} \cap r_2^{\mathcal{I}}$ (we use it for a similar purpose as [Gli+08]). We let $\mathsf{inv}(r_1 \sqcap r_2) = \mathsf{inv}(r_1) \sqcap \mathsf{inv}(r_2)$ and assume w.l.o.g. that for each role inclusions $r \sqsubseteq s$ of an Horn-$\mathcal{ALCHIQ}^{\sqcap}$ TBox $\mathcal{T}$, (i) $\mathsf{inv}(r) \sqsubseteq \mathsf{inv}(s) \in \mathcal{T}$, and (ii) $s \in \{p, p^-\}$ for a role name p. For a set W and a concept or role conjunction $\Gamma = \gamma_1 \sqcap \ldots \sqcap \gamma_m$, we write $\Gamma \subseteq W$ for $\{\gamma_1, \ldots, \gamma_m\} \subseteq W$.

It is handy to eliminate transitivity axioms from $\mathcal{SHIQ}$ TBoxes (see, e.g.,[HMS07]). We use the transformation from [Kaz09a], which ensures the resulting TBox is in normal form.

Definition 4.31. *Let $\mathcal{T}^*$ be the Horn-$\mathcal{ALCHIQ}$ TBox obtained from a Horn-$\mathcal{SHIQ}$ TBox $\mathcal{T}$ by (i) adding for every $A \sqsubseteq \forall s.B \in \mathcal{T}$ and every transitive role r with $r \sqsubseteq^*_{\mathcal{T}} s$, the axioms $A \sqsubseteq \forall r.B^r$, $B^r \sqsubseteq \forall r.B^r$ and $B^r \sqsubseteq B$, where B^r is a fresh concept name; and (ii) removing all transitivity axioms.*

The transformation does not preserve answers to CQs where non-simple roles occur. However, we can relax the notion of match and then use the translated TBox for answering arbitrary CQs.

Definition 4.32. *Let $\mathcal{T}$ be a Horn-$\mathcal{SHIQ}$ TBox. A $\mathcal{T}$-*match *for a query q in an interpretation $\mathcal{I}$ is a mapping π from variables of q to elements in $\Delta^{\mathcal{I}}$ that satisfies the following:*

(a) If $\alpha = p(\vec{t})$ is a body atom in q, where $p \in \mathsf{N_C}$ or p is a simple role in $\mathcal{T}$, then $\pi(\vec{t}) \in p^{\mathcal{I}}$.

*(b) If $\alpha = s(x, y)$ with s non-simple, then there exist a transitive $r \sqsubseteq^*_{\mathcal{T}} s$ and $d_1 \in \Delta^{\mathcal{I}}, \ldots, d_k \in \Delta^{\mathcal{I}}$ such that $d_1 = \pi(x)$, $d_k = \pi(y)$, and $(d_i, d_{i+1}) \in r^{\mathcal{T}}$ for all each $1 \leq i < k$; we call this sequence $d_1 \in \Delta^{\mathcal{I}}, \ldots, d_k \in \Delta^{\mathcal{I}}$ an r-*path from $\pi(x)$ to $\pi(y)$.

The set $ans^{\mathcal{T}}(\mathcal{O}, q)$ is defined as $ans(\mathcal{O}, q)$ but using $\mathcal{T}$-matches instead of matches. The next characterization follows from known techniques, see e.g. [EOS12] for a similar result.

Proposition 4.33. *For any Horn-$\mathcal{SHIQ}$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ and CQ q, we have $ans(\mathcal{O}, q) = ans^{\mathcal{T}}((\mathcal{T}^*, \mathcal{A}), q)$.*

4.4.2 Canonical Models

A stepping stone for the tailored query answering methods for Horn DLs and languages like $\mathsf{Datalog}^{\pm}$ is the *canonical model property* [Eit+08c; ORS11; CGL09]. In particular, for a consistent Horn-$\mathcal{ALCHIQ}^{\sqcap}$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, there exists a model $\mathcal{I}$ of $\mathcal{O}$ that can be homomorphically embedded into any other model $\mathcal{I}'$ of $\mathcal{O}$. We show that such an $\mathcal{I}$ can be built in three steps:

(1) Close $\mathcal{T}$ under specially tailored inferences rules.
(2) Close $\mathcal{A}$ under all but existential axioms of $\mathcal{T}$.
(3) Extend $\mathcal{A}$ by "applying" the existential axioms of $\mathcal{T}$.

For Step (1), we tailor from the inference rules in [Kaz09a; ORS10] a calculus to support model building for Horn-$\mathcal{ALCHIQ}^{\sqcap}$ ontologies.

$$\frac{M \sqsubseteq \exists S.(N \sqcap N') \quad N \sqsubseteq A}{M \sqsubseteq \exists S.(N \sqcap N' \sqcap A)}\ \mathbf{R}^{c}_{\sqsubseteq} \qquad \frac{M \sqsubseteq \exists (S \sqcap S').N \quad S \sqsubseteq r}{M \sqsubseteq \exists (S \sqcap S' \sqcap r).N}\ \mathbf{R}^{r}_{\sqsubseteq} \qquad \frac{M \sqsubseteq \exists S.N \sqcap \bot}{M \sqsubseteq \bot}\ \mathbf{R}_{\bot}$$

$$\frac{M \sqsubseteq \exists (S \sqcap r).N \quad A \sqsubseteq \forall r.B}{M \sqcap A \sqsubseteq \exists (S \sqcap r).(N \sqcap B)}\ \mathbf{R}_{\forall} \qquad \frac{M \sqsubseteq \exists (S \sqcap \mathsf{inv}(r)).(N \sqcap A) \quad A \sqsubseteq \forall r.B}{M \sqsubseteq B}\ \mathbf{R}^{-}_{\forall}$$

$$\frac{M \sqsubseteq \exists (S \sqcap r).(N \sqcap B) \quad A \sqsubseteq \leqslant 1\, r.B \quad M' \sqsubseteq \exists (S' \sqcap r).(N' \sqcap B)}{M \sqcap M' \sqcap A \sqsubseteq \exists (S \sqcap S' \sqcap r).(N \sqcap N' \sqcap B)}\ \mathbf{R}_{\leq}$$

$$\frac{M \sqsubseteq \exists (S \sqcap \mathsf{inv}(r)).(N_1 \sqcap N_2 \sqcap A) \quad A \sqsubseteq \leqslant 1\, r.B \quad N_1 \sqcap A \sqsubseteq \exists (S' \sqcap r).(N' \sqcap B \sqcap C)}{M \sqcap B \sqsubseteq C \quad M \sqcap B \sqsubseteq \exists (S \sqcap \mathsf{inv}(S' \sqcap r)).(N_1 \sqcap N_2 \sqcap A)}\ \mathbf{R}^{-}_{\leq}$$

Table 4.4: Inference rules. $M^{(\prime)}$, $N^{(\prime)}$, (resp., $S^{(\prime)}$) are conjunctions of atomic concepts (roles); A, B are atomic concepts

$$\begin{aligned}
B(y) &\leftarrow A(x), r(x,y) \ \text{ for each } A \sqsubseteq \forall r.B \in \mathcal{T} \\
B(x) &\leftarrow A_1(x), \ldots, A_n(x) \ \text{ for all } A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B \in \Xi(\mathcal{T}) \\
r(x,z) &\leftarrow r(x,y), r(y,z) \ \text{ for all transitive roles } r \text{ in } \mathcal{T} \\
r(x,y) &\leftarrow r_1(x,y), \ldots, r_n(x,y) \ \text{ for all } r_1 \sqcap \ldots \sqcap r_n \sqsubseteq r \in \mathcal{T} \\
\bot(x) &\leftarrow A(x), r(x,y_1), r(x,y_2), B(y_1), B(y_2), y_1 \neq y_2 \ \text{ for each } A \sqsubseteq \leqslant 1\, r.B \in \mathcal{T} \\
\Gamma &\leftarrow A(x), A_1(x), \ldots, A_n(x), r(x,y), B(y) \\
&\quad \text{for all } A_1 \sqcap \ldots \sqcap A_n \sqsubseteq \exists (r_1 \sqcap \ldots \sqcap r_m).(B_1 \sqcap \ldots \sqcap B_k) \\
&\quad \text{and } A \sqsubseteq \leqslant 1\, r.B \text{ of } \Xi(\mathcal{T}) \text{ such that } r = r_i \text{ and } B = B_j \text{ for some } i, j \\
&\quad \text{with } \Gamma \in \{B_1(y), \ldots, B_k(y), r_1(x,y), \ldots, r_k(x,y)\}
\end{aligned}$$

Table 4.5: (Completion rules) Datalog program $\mathsf{cr}(\mathcal{T})$.

Definition 4.34. *Given a Horn-$\mathcal{ALCHIQ}^{\sqcap}$ TBox $\mathcal{T}$, $\Xi(\mathcal{T})$ is the TBox obtained from $\mathcal{T}$ by exhaustively applying the inference rules in Table 4.4.*

For Step (2), we use Datalog rules that express the semantics of GCIs, ignoring existential axioms.

Definition 4.35. *Given a Horn-$\mathcal{ALCHIQ}^{\sqcap}$ TBox $\mathcal{T}$, $\mathsf{cr}(\mathcal{T})$ is the* Datalog *program described in Table 4.5.*

Given a consistent Horn-$\mathcal{ALCHIQ}^{\sqcap}$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, the least model $\mathcal{J}$ of the Datalog program $\mathsf{cr}(\mathcal{T}) \cup \mathcal{A}$ is almost a canonical model of $\mathcal{O}$; however, existential axioms may be violated. We deal with this in Step (3), by extending $\mathcal{J}$ with new domain elements as required by axioms $A \sqsubseteq \exists r.N$ in $\Xi(\mathcal{T})$, akin to database *chase* [MM79] where fresh values and tuples are introduced to satisfy the given dependencies.

Definition 4.36. *Let $\mathcal{T}$ be a Horn-$\mathcal{ALCHIQ}^{\sqcap}$ TBox and $\mathcal{I}$ an interpretation. A GCI $M \sqsubseteq \exists S.N$ is* applicable at $e \in \Delta^{\mathcal{I}}$ *if*

(a) $e \in M^{\mathcal{I}}$,

(b) there is no $e' \in \Delta^{\mathcal{I}}$ *with* $(e, e') \in S^{\mathcal{I}}$ *and* $e' \in N^{\mathcal{I}}$*,*

(c) there is no axiom $M' \sqsubseteq \exists S'.N' \in \mathcal{T}$ *such that* $e \in (M')^{\mathcal{I}}$*,* $S \subseteq S'$*,* $N \subseteq N'$*, and* $S \subset S'$ *or* $N \subset N'$*.*

An interpretation $\mathcal{J}$ *obtained from* $\mathcal{I}$ *by an* application *of an applicable axiom* $M \sqsubseteq \exists S.N$ *at* $e \in \Delta^{\mathcal{I}}$ *is defined as:*

- $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}} \cup \{d\}$ *with* d *a new element not present in* $\Delta^{\mathcal{I}}$ *(we call* d *a* successor *of* e*),*

- *For each atomic* $A \in \mathsf{N_C}$ *and* $o \in \Delta^{\mathcal{J}}$*, we have* $o \in A^{\mathcal{J}}$ *if a)* $o \in \Delta^{\mathcal{I}}$ *and* $o \in A^{\mathcal{I}}$*; or b)* $o = d$ *and* $A \in N$*.*

- *For each role name* r *and* $o, o' \in \Delta^{\mathcal{J}}$*, we have* $(o, o') \in r^{\mathcal{J}}$ *if a)* $o, o' \in \Delta^{\mathcal{I}}$ *and* $(o, o') \in r^{\mathcal{I}}$*; or b)* $(o, o') = (e, d)$ *and* $r \in S$*; or c)* $(o, o') = (d, e)$ *and* $\mathsf{inv}(r) \in S$*.*

We denote by chase$(\mathcal{I}, \mathcal{T})$ *a possibly infinite interpretation obtained from* $\mathcal{I}$ *by applying the existential axioms in* $\mathcal{T}$*. We require the application to* fair*: the application of an applicable axiom can not be infinitely postponed.*

We note that *chase*$(\mathcal{I}, \mathcal{T})$ is unique up to renaming of domain elements. As usual in DLs, it can be seen as a 'forest': application of existential axioms simply attaches 'trees' to a possibly arbitrarily shaped $\mathcal{I}$. The following statement can be shown similarly as in [ORS11].

Proposition 4.37. *Let* $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ *be a Horn-*$\mathcal{ALCHIQ}^{\sqcap}$ *ontology. Then* $\mathcal{O}$ *is consistent iff* $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$ *consistent. Moreover, if* $\mathcal{O}$ *is consistent, then*

(a) chase$(MM(\mathcal{A} \cup \mathsf{cr}(\mathcal{T})), \Xi(\mathcal{T}))$ *is a model of* $\mathcal{O}$*, and*

(b) chase$(MM(\mathcal{A} \cup \mathsf{cr}(\mathcal{T})), \Xi(\mathcal{T}))$ *can be homomorphically embedded into any model of* $\mathcal{O}$*.*

Proof. Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be a Horn-$\mathcal{ALCHIQ}^{\sqcap}$ ontology.

Suppose $\mathcal{O}$ is consistent and $\mathcal{J}$ is a model of $\mathcal{O}$. We first show that $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$ is consistent, and afterwards show (a) and (b). Due to the UNA, we can w.l.o.g. assume that $a^{\mathcal{J}} = a$ for each constant $a \in \mathsf{N_I}$. A model of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$ can built by simply restricting the domain of $\mathcal{J}$ to constants. Let $\mathcal{J}'$ be the interpretation such that

- $\Delta^{\mathcal{J}'} = \mathsf{N_I}$;

- $A^{\mathcal{J}'} = A^{\mathcal{J}} \cap \Delta^{\mathcal{J}'}$ and $r^{\mathcal{J}'} = r^{\mathcal{J}} \cap \Delta^{\mathcal{J}'} \times \Delta^{\mathcal{J}'}$, for all concepts names A and role names r.

$\mathcal{J}'$ is a model of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$ because $\mathcal{J}$ is a model of $\mathcal{T}$ and since all axioms in $\Xi(\mathcal{T})$ are logical consequences of $\mathcal{T}$.

Assume the least model $\mathcal{I}_{\mathcal{A}}$ of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$, which exists due to consistency $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$. Let $\mathcal{I}_{\mathcal{O}} = \mathit{chase}(\mathcal{I}_{\mathcal{A}}, \Xi(\mathcal{T}))$. We show next that $\mathcal{I}_{\mathcal{O}}$ is a model of $\mathcal{O}$, i.e. show (a). To show the statement we need some book-keeping when building $\mathcal{I}_{\mathcal{O}}$. We assume $\Delta^{\mathcal{I}_{\mathcal{A}}} = \mathsf{N_I}$ and prescribe the naming of fresh domain elements introduced during the chase procedure. In particular, if d is a successor of e according to Definition 4.36, then d is an expression of the form $e \cdot n$, where n is a integer. We show that $\mathcal{I}_{\mathcal{O}}$ satisfies each statement in $\mathcal{O}$:

(1) For assertions $A(a) \in \mathcal{A}$ and $r(a,b) \in \mathcal{A}$, we have $a^{\mathcal{I}_\mathcal{A}} \in A^{\mathcal{I}_\mathcal{A}}$ and $(a^{\mathcal{I}_\mathcal{A}}, b^{\mathcal{I}_\mathcal{A}}) \in r^{\mathcal{I}_\mathcal{A}}$ because $\mathcal{I}_\mathcal{A}$ is a model of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$. We have $a^{\mathcal{I}_\mathcal{O}} \in A^{\mathcal{I}_\mathcal{O}}$ and $(a^{\mathcal{I}_\mathcal{O}}, b^{\mathcal{I}_\mathcal{O}}) \in r^{\mathcal{I}_\mathcal{O}}$ because $\mathcal{I}_\mathcal{O}$ is an extension $\mathcal{I}_\mathcal{A}$ by construction.

(2) Assume an axiom $M \sqsubseteq B \in \Xi(\mathcal{T})$, where M is a conjunction of atomic concepts, and also assume a domain element $e \in M^{\mathcal{I}_\mathcal{O}}$. Note that $\mathcal{T} \subseteq \Xi(\mathcal{T})$. If $e \in \mathsf{N_I}$, then $e \in B^{\mathcal{I}_\mathcal{O}}$ since $\mathcal{I}_\mathcal{A}$ is a model of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$. Assume $e = w \cdot n$. We know e is a successor of w introduced in $\mathcal{I}_\mathcal{O}$ by an application of some $M' \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$. By the construction of $\mathcal{I}_\mathcal{O}$, e satisfies exactly the atomic concepts in N. It remains to see that $B \in N$. This follows from the inference rule ($\mathbf{R}^c_\sqsubseteq$). Indeed, if $B \notin N$, then we can apply ($\mathbf{R}^c_\sqsubseteq$) to obtain the axiom $M' \sqsubseteq \exists S.N \sqcap B \in \Xi(\mathcal{T})$. This makes $M' \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ inapplicable at e due a violation of (c) in Definition 4.36.

(3) To show that existential axioms are satisfied, first take an arbitrary domain element $e \in A^{\mathcal{I}_\mathcal{O}}$. We say $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ is *relevant for* e if there is no axiom $M' \sqsubseteq \exists S'.N' \in \mathcal{T}$ such that $e \in (M')^{\mathcal{I}}$, $S \subseteq S'$, $N \subseteq N'$, and $S \subset S'$ or $N \subset N'$. To prove that $\mathcal{I}_\mathcal{O}$ satisfies each existential axiom of $\Xi(\mathcal{T})$, it suffices to show that $\mathcal{I}_\mathcal{O}$ satisfies each existential axiom that is relevant for e. To this end, assume $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ relevant for e. Suppose $e \in M^{\mathcal{I}_\mathcal{O}}$ and $e \notin (\exists S.N)^{\mathcal{I}_\mathcal{O}}$. Then $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ is applicable in $\mathcal{I}_\mathcal{O}$ at e according to Definition 4.36. This leads to a contradiction: by the fairness of chase, the axiom $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ must be applied and thus $e \in (\exists S.N)^{\mathcal{I}_\mathcal{O}}$.

(4) Assume an axiom $A \sqsubseteq \forall r.B \in \mathcal{T}$ and a domain element $e \in A^{\mathcal{I}_\mathcal{O}}$. Suppose there is $e' \in \Delta^{\mathcal{I}_\mathcal{O}}$ such that $(e, e') \in r^{\mathcal{I}_\mathcal{O}}$ and $e' \notin B^{\mathcal{I}_\mathcal{O}}$. Due to the definition of $\mathcal{I}_\mathcal{O}$, we have 3 possible cases:

  (i) $e, e' \in \mathsf{N_I}$ and $(e, e') \in r^{\mathcal{I}_\mathcal{A}}$. We have that $e' \in B^{\mathcal{I}_\mathcal{O}}$ because $\mathcal{I}_\mathcal{A}$ is a model of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$ by assumption.

  (ii) $e' = e \cdot n$ for some integer n, where e' was introduced by applying some axiom $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$. Note that, by the construction of $\mathcal{I}_\mathcal{O}$, we must have $e \in M^{\mathcal{I}_\mathcal{O}}$ and $r \in S$. From the inference rule ($\mathbf{R}_\forall$) we know that $M \sqcap A \sqsubseteq \exists S.(N \sqcap B) \in \Xi(\mathcal{T})$. We know that $e \in (M \sqcap A)^{\mathcal{I}_\mathcal{O}}$. Then due to maximality of $M \sqsubseteq \exists S.N$ at e, we have $N \sqcap B = N$, i.e. $B \in N$. By the construction of $\mathcal{I}_\mathcal{O}$, $e' \in B^{\mathcal{I}_\mathcal{O}}$.

  (iii) $e = e' \cdot n$ for some integer n, where e was introduced by applying some axiom $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$. By the construction of $\mathcal{I}_\mathcal{O}$, we have $r^- \in S$ and $A \in N$. Then by the inference rule ($\mathbf{R}^-_\forall$), we have $M \sqsubseteq B \in \Xi(\mathcal{T})$. We have already shown above that $\mathcal{I}_\mathcal{O}$ satisfies $M \sqsubseteq B$. Since $e' \in M^{\mathcal{I}_\mathcal{O}}$ by the construction of $\mathcal{I}_\mathcal{O}$, we have $e' \in A^{\mathcal{I}_\mathcal{O}}$.

(5) Assume a role inclusion $S \sqsubseteq r \in \Xi(\mathcal{T})$ and a pair $(e, e') \in S^{\mathcal{I}_\mathcal{O}}$. Due to the definition of $\mathcal{I}_\mathcal{O}$, we have 2 possible cases:

  (i) $e, e' \in \mathsf{N_I}$. Then $(e, e') \in r^{\mathcal{I}_\mathcal{O}}$ because $\mathcal{I}_\mathcal{A}$ is a model of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$ by assumption.

  (ii) $e' = e \cdot n$ for some integer n, where e' was introduced by applying some axiom $M \sqsubseteq \exists S'.N \in \Xi(\mathcal{T})$ with $S \subseteq S'$. We know from the inference rule ($\mathbf{R}^r_\sqsubseteq$) that $M \sqsubseteq \exists S' \sqcap r.N \in \Xi(\mathcal{T})$. Due to maximality of $M \sqsubseteq \exists S'.N$, we must have $S' \sqcap r = S'$, which implies $(e, e') \in r^{\mathcal{I}_\mathcal{O}}$.

(iii) $e = e' \cdot n$ for some integer n, where e was introduced by applying some axiom $M \sqsubseteq \exists S'.N \in \Xi(\mathcal{T})$ with $S^- \subseteq S'$. Note that $S^- \sqsubseteq r^- \in \mathcal{T}$ (see preliminaries). We know from the inference rule ($\mathbf{R}^r_{\sqsubseteq}$) that $M \sqsubseteq \exists S' \sqcap r^-.N \in \Xi(\mathcal{T})$. Again, due to maximality of $M \sqsubseteq \exists S'.N$, we must have $S' \sqcap r^- = S'$, which implies $(e', e) \in (r^-)^{\mathcal{I}_\mathcal{O}}$ and $(e, e') \in r^{\mathcal{I}_\mathcal{O}}$.

(6) Assume an axiom $A \sqsubseteq \leqslant 1\, r.B \in \mathcal{T}$ and a domain element $e \in A^{\mathcal{I}_\mathcal{O}}$. Suppose there is $e_1, e_2 \in \Delta^{\mathcal{I}_\mathcal{O}}$ such that $e_1 \neq e_2$, $\{ (e, e_1), (e, e_2) \} \subseteq r^{\mathcal{I}_\mathcal{O}}$ and $\{ e_1, e_2 \} \subseteq B^{\mathcal{I}_\mathcal{O}}$. We have the following possible cases:

(i) $\{ e_1, e_2 \} \subseteq \mathsf{N_I}$. Then by the construction of $\mathcal{I}_\mathcal{O}$ we must have $e \in \mathsf{N_I}$. We arrive at a contradiction to the assumption that $\mathcal{I}_\mathcal{A}$ is a model of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$; the constraint representing $A \sqsubseteq \leqslant 1\, r.B \in \mathcal{T}$ must be violated.

(ii) $e_1, e \in \mathsf{N_I}$ and e_2 is of the form $e_2 = e \cdot n$ for some integer. Assume e_2 was introduced by applying an applicable axiom $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ at e. Note we have $e \in M^{\mathcal{I}_\mathcal{O}}$. By a rule of the last type in Table 4.5, we have that $e_1 \in N^{\mathcal{I}_\mathcal{A}}$ and $(e, e_1) \in S^{\mathcal{I}_\mathcal{A}}$. This shows that $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ was never applicable at e. Contradiction.

(iii) $e_2, e \in \mathsf{N_I}$ and e_1 is of the form $e_1 = e \cdot n$ for some integer. Symmetric to the above.

(iv) e_1, e_2 are of the form $e_1 = e \cdot n$ and $e_2 = e \cdot n'$. Suppose e_1, e_2 where introduced by applying axioms $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ and $M' \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ at e. Then by the construction of $\mathcal{I}_\mathcal{O}$ we have $r \in S$, $r \in S'$, $B \in N$ and $B \in N'$. Then by the inference rule ($\mathbf{R}_{\leqslant}$), we have $M \sqcap M' \sqcap A \sqsubseteq \exists (S \sqcap S').(N \sqcap N') \in \Xi(\mathcal{T})$. Since $e \in (M \sqcap M' \sqcap A)^{\mathcal{I}_\mathcal{O}}$, we have a violation of applicability of $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ and $M' \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ at e, i.e. they are not maximal.

(v) $e = e_1 \cdot n$ and $e_2 = e \cdot n'$ obtained by applying some axioms $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ and $M' \sqsubseteq \exists S'.N' \in \Xi(\mathcal{T})$ at e_1 and e, respectively. By the construction of $\mathcal{I}_\mathcal{O}$, we have have $A \in N$, $r^- \in S$, $r \in S'$ and $B \in N'$. Then by the inference rule ($\mathbf{R}^-_{\leqslant}$), we have $M \sqcap B \sqsubseteq C \in \Xi(\mathcal{T})$ for all $C \in N'$ and also $M \sqcap B \sqsubseteq \exists (S \sqcap (S')^-).\bar{N} \in \Xi(\mathcal{T})$. Since $e_1 \in (M \sqcap B)^{\mathcal{I}_\mathcal{O}}$, we have $(S^-)^- \subset S$ by the maximality of $M \sqsubseteq \exists S.N$. Due to point (2) in this proof, we also have $e_1 \in C^{\mathcal{I}_\mathcal{O}}$ for all $C \in N'$. This shows that $M' \sqsubseteq \exists S'.N' \in \Xi(\mathcal{T})$ was not applicable at e, i.e. maximality violated.

(7) It remains to see that $\bot^{\mathcal{I}_\mathcal{O}} = \emptyset$. First note that $\mathsf{N_I} \cap \bot^{\mathcal{I}_\mathcal{O}} = \emptyset$ because $\mathcal{I}_\mathcal{A}$ is a model of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$. Thus it suffices to prove the following statement: if $e \cdot n \in \bot^{\mathcal{I}_\mathcal{O}}$, then also $e \in \bot^{\mathcal{I}_\mathcal{O}}$. Assume some $e \cdot n \in \bot^{\mathcal{I}_\mathcal{O}}$. Suppose $e \cdot n$ was introduced by applying an axiom $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$. Then by the definition of $\mathcal{I}_\mathcal{O}$, $\bot \in N$. By the inference rule ($\mathbf{R}_\bot$), we have $M \sqsubseteq \bot \in \Xi(\mathcal{T})$. Since $e \in M^{\mathcal{I}_\mathcal{O}}$, by point (2) in this proof we have $e \in \bot^{\mathcal{I}_\mathcal{O}}$.

It remains to see (b), i.e. that $\mathcal{I}_\mathcal{O}$ can be homomorphically embedded into any model $\mathcal{I}$ of $\mathcal{O}$. A homomorphism h from $\mathcal{I}_\mathcal{O}$ to $\mathcal{I}$ can be inductively defined as follows:

- $h(e) = e^{\mathcal{I}}$ for all $e \in \mathsf{N_I} \cap \Delta^{\mathcal{I}_\mathcal{O}}$. It is straightforward to see that $e_1 \in A^{\mathcal{I}_\mathcal{O}}$ and $(e_1, e_2) \in r^{\mathcal{I}_\mathcal{O}}$ imply $e_1 \in A^{\mathcal{I}}$ and $(e_1, e_2) \in r^{\mathcal{I}}$ for all $e_1, e_2 \in \mathsf{N_I}$, concepts A and roles r.

- Assume $e \cdot n \in \Delta^{\mathcal{I}_\mathcal{O}}$ was introduced in $\mathcal{I}_\mathcal{O}$ by an application of $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$. Note that $e \in M^{\mathcal{I}_\mathcal{O}}$. It suffices to define $h(e \cdot n) = e'$ where $e' \in \Delta^{\mathcal{I}}$ is an element such that $S \subseteq \{\ r \mid (h(e), e') \in r^{\mathcal{I}}\ \}$ and $N \subseteq \{\ A \mid e' \in A^{\mathcal{I}}\ \}$. Note that such e' exists. Indeed, by the induction hypothesis, $h(e) \in M^{\mathcal{I}}$. Since $\mathcal{I}$ is a model of $\Xi(\mathcal{T})$, we must have $h(e) \in (\exists S.N)^{\mathcal{I}}$.

It remains to show that consistency of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$ implies consistency of $\mathcal{O}$. Assume $\mathcal{O}$ is inconsistent and suppose $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$ is consistent. Then there exists the least model $\mathcal{I}_\mathcal{A}$ of $\mathcal{A} \cup \mathsf{cr}(\mathcal{T})$, and thus $\mathcal{I}_\mathcal{O} = \mathit{chase}(MM(\mathcal{A} \cup \mathsf{cr}(\mathcal{T})), \Xi(\mathcal{T}))$ is defined. As we shown in (a), $\mathcal{I}_\mathcal{O} \models \mathcal{O}$. Contradiction. □

In database terms, this means that (1) checking consistency of $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ reduces to evaluating the (plain) Datalog query $\mathsf{cr}(\mathcal{T})$ over the database $\mathcal{A}$; (2) answering concept query $C(a)$ over $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ reduces to the boolean Datalog query $\{q \leftarrow C(a)\} \cup \mathsf{cr}(\mathcal{T})$ over $\mathcal{A}$.

Note that $\Xi(\mathcal{T})$ can be computed in exponential time in size of $\mathcal{T}$: the calculus only infers axioms of the form $M \sqsubseteq B$ and $M \sqsubseteq \exists S.N$, where M, N are conjunctions of atomic concepts, B is atomic and S is a conjunction of roles. The number of such axiom is single exponential in the size of $\mathcal{T}$.

Example 4.38. *Let $\Sigma = (\mathcal{T}, \mathcal{A})$ be a Horn-$\mathcal{SHIQ}$ ontology, where*

$$\mathcal{T} = \left\{ \begin{array}{llll} (t_1) & A_1 \sqsubseteq \exists R.A_2, & (t_2) & B_1 \sqsubseteq \forall R.B_2, \\ (t_3) & A_2 \sqcap B_2 \sqsubseteq C, & (t_4) & \exists R.C \sqsubseteq D \end{array} \right\},$$

and $\mathcal{A} = \{A_1(a),\ B_1(a)\}$. It is not difficult to see $\Sigma \models D(a)$. In the following we show how to reduce this instance query to a Datalog *query over $\mathcal{A}$.*

First, the TBox axiom (t_4) is normalized to

$$(t_4') \quad C \sqsubseteq \forall R^{-}.D$$

Next, $\mathcal{T}$ is saturated as following:

$$\begin{array}{lll} (t_5) & A_1 \sqcap B_1 \sqsubseteq \exists R.(A_2 \sqcap B_2) & \mathbf{R}_\forall(t_1, t_2) \\ (t_6) & A_1 \sqcap B_1 \sqsubseteq \exists R.(A_2 \sqcap B_2 \sqcap C) & \mathbf{R}^c_\sqsubseteq(t_3, t_5) \\ (t_7) & A_1 \sqcap B_1 \sqsubseteq C & \mathbf{R}^{-}_\forall(t_4', t_6) \end{array}$$

Then the completion rules $cr(\mathcal{T})$ are

$$\begin{array}{ll} B_2(X) \leftarrow B_1(X), R(X, Y) & (t_2) \\ C(X) \leftarrow A_2(X), B_2(X) & (t_3) \\ D(X) \leftarrow R(X, Y), C(Y) & (t_4') \\ C(X) \leftarrow A_1(X), B_1(X) & (t_7) \end{array}$$

Finally, we can check that $cr(\mathcal{T}) \cup \mathcal{A} \models D(a)$ holds as we expected.

4.5 Go Beyond Instance Queries

So far in this chapter we have only considered dl-atoms with positive instance queries of concept or role names. Recall that a dl-query $Q(\mathbf{t})$ is either

(a) a concept inclusion axiom F or its negation $\neg F$; or

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

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

(d) of the forms $= (t_1, t_2)$ or $\neq (t_1, t_2)$, where t_1 and t_2 are terms.

We show how to reduce them to dl-programs with only positive concept assertions.

Positive Concept Inclusions

It is well-known that for any DL ontology L, the concept inclusion $L \models C \sqsubseteq D$ holds iff $L \cup \{C(\tau)\} \models D(\tau)$, where τ is a fresh constant. We can use this property to reduce the dl-atoms with concept inclusion to those with concept instance queries.

Note that if the UNA is assumed in L, the above statement may not hold because of the cardinality of the domain. For instance, let $L = \{\leq 2U.\top \sqsubseteq \bot, A(a)\}$, where U is the universal role and concept names $\{A, B\} \subseteq \mathsf{N_C}$. By default, if the UNA is assumed, the domain of L has only one element $\Delta^{\mathcal{I}} = \mathsf{N_I} = \{a\}$. Clearly $L \not\models A \sqsubseteq B$. However, $L \cup A(\tau)$ will trigger $\leq 2U.\top \sqsubseteq \bot$, infer that $L \cup A(\tau)$ is inconsistent, and conclude $L \cup A(\tau) \models B(\tau)$. To exclude this case, we always assume a large enough domain in the DL ontology.

Definition 4.39. *Let $\mathcal{KB} = (L, P)$ be a dl-program, we define a positive concept inclusion reduction, denoted Ψ_{pci}, by rewriting all dl-atoms $DL[\lambda; C \sqsubseteq D]()$ in P to $DL[\lambda, C \uplus C_\lambda; D_\lambda](o_{C_\lambda})$ and adding facts $C_\lambda(o_{C_\lambda})$, where C_λ is a fresh concept and o_{C_λ} is a fresh individual.*

Proposition 4.40. *There is a 1-1 correspondence between the models of $\mathcal{KB} = (L, P)$ and $\Psi_{pci}(\mathcal{KB})$ under both well-founded semantics and stable model semantics:*

- *Let a be a ground atom from HB_P. Then, $\mathcal{KB} \models^{wf} a$ iff $\Psi_{pci}(\mathcal{KB}) \models^{wf} a$.*
- *Every answer set of $\mathcal{KB}$ is expendable to an answer set of $\Psi_{pci}(\mathcal{KB})$; and*
- *for every answer set J of $\Psi_{pci}(\mathcal{KB})$, its restriction $I = J|_{HB_P}$ to HB_P is an answer set of $\mathcal{KB}$.*

Proof. We note that the interpretation I of $\mathcal{KB}$ can be seen as a set of Datalog facts, and we slightly abuse the notion of Ψ_{pci}, such that

$$\Psi_{pci}(I) = I \cup \{C_\lambda(o_{C_\lambda}) \mid DL[\lambda; C \sqsubseteq D]() \text{ occurs in } P\}$$

To prove this proposition, we need the following two lemmas.

□

Lemma 4.41. $I \models DL[\lambda; C \sqsubseteq D]()$ *iff* $I \cup \{C_\lambda(o_{C_\lambda})\} \models DL[\lambda, C \uplus C_\lambda; D_\lambda](o_{C_\lambda})$

Proof.

$$\begin{array}{ll} & I \models DL[\lambda; C \sqsubseteq D]() \\ \text{iff} & L(I;\lambda) \models C \sqsubseteq D \\ \text{iff} & L(I;\lambda) \cup \{C(o_{C_\lambda})\} \models D(o_{C_\lambda}) \\ \text{iff} & L(I \cup \{C_\lambda(o_{C_\lambda})\}; \lambda, C \uplus C_\lambda) \models D(o_{C_\lambda}) \\ \text{iff} & I \cup \{C_\lambda(o_{C_\lambda})\} \models DL[\lambda, C \uplus C_\lambda; D_\lambda](o_{C_\lambda}) \end{array}$$

□

Recall that $\gamma_{\mathcal{KB}}(I) = MM(\mathcal{KB}^I) = T^\infty_{\mathcal{KB}}(\mathcal{KB}^I)$, where $T_{\mathcal{KB}}$ is the immediate consequence operator.

Lemma 4.42. *Let* $\mathcal{KB} = (L, P)$ *be a dl-program, then* $\gamma_{\mathcal{KB}}(I) = \gamma_{\Psi_{pci}(\mathcal{KB})}(\Psi_{pci}(I))|_{HB_P}$

Proof. This can be proved inductively on the number n of steps of $T_{\mathcal{KB}}$ needed for computing $\gamma_{\mathcal{KB}}$ using Lemma 4.41. □

Based on the above lemma and using the same techniques of the proof of Theorem 4.7 and Theorem 4.9, we complete the proof of Proposition 4.40.[4]

Example 4.43. *Let dl-program* $\mathcal{KB} = (L, P)$*, where* $L = (\mathcal{T}, \emptyset)$*,* $\mathcal{T} = \{C \sqsubseteq D, D \sqsubseteq E\}$*,* $P = \{q \leftarrow DL[; C \sqsubseteq E](), not\ DL[; E \sqsubseteq C]().\}$

Then $\Psi_{pci}(\mathcal{KB}) = (L, P')$*, where*

$$P' = \left\{ \begin{array}{l} q \leftarrow DL[C \uplus C_{\lambda_0}; E](o_{C_{\lambda_0}}), not\ DL[E \uplus E_{\lambda_0}; C](o_{E_{\lambda_0}}). \\ \quad C_{\lambda_0}(o_{C_{\lambda_0}}). \quad E_{\lambda_0}(o_{E_{\lambda_0}}). \end{array} \right\}$$

It easy to check that $\mathcal{KB}$ *has a single answer set* $\{q\}$*, and* $\Psi_{pci}(KB)$ *has a single answer set* $\{q, C_{\lambda_0}(o_{C_{\lambda_0}}), E_{\lambda_0}(o_{E_{\lambda_0}})\}$*. The correspondence is as expected.*

Negative Concept Inclusion, Negative Concept and Role Assertions

The reasoning tasks negative concept inclusion, negative Concept and role assertions can be reduced to instance queries.

- The negated concept inclusion axiom $\neg(C \sqsubseteq D)$ is equivalent to the concept membership axiom $(C \sqcap \neg D)(\tau)$ (where τ is a fresh individual),
- The negated concept membership axiom $\neg(C(a))$ is equivalent to the concept membership axiom $(\neg C)(a)$.

[4] We skip the details of the proof here. Basically, if one changes all the Φ to Φ_{pci} in the proofs of Lemma 4.6, Theorem 4.7 and Theorem 4.9, then we obtain a proof of Proposition 4.40.

- The negated abstract role membership axiom $\{\neg R(a,b)\}$ in a DL knowledge base L can be removed by using that $L \cup \{\neg R(a,b)\}$ is unsatisfiable iff $L \cup \{A(a), B(b), \exists R.B \sqsubseteq \neg A\}$ is unsatisfiable (where A and B are two fresh atomic concepts and L is any DL knowledge base).

Therefore, dl-programs with dl-atoms in the form of these axioms can be reduced to the dl-programs with only concept assertion queries similarly as the techniques used for reducing positive concept inclusion.

Equalities

Recall that for the simplicity of Datalog reduction, we assumed that $\mathcal{LDL}^+$ and Horn-$\mathcal{SHIQ}$ are under unique name assumption (UNA) throughout this chapter. However, this is not a real restriction. In fact, UNA can be dropped from $\mathcal{LDL}^+$ and Horn-$\mathcal{SHIQ}$ by adding additional standard rules for reflexive, symmetric and transitive properties of the equality predicate (eq):

$$eq(X,X) \leftarrow \top(X). \tag{4.24}$$
$$eq(X,Y) \leftarrow eq(Y,X). \tag{4.25}$$
$$eq(X,Z) \leftarrow eq(X,Y), eq(Y,Z). \tag{4.26}$$

Then by replacing the atoms $\mathbf{t_1} = \mathbf{t_2}$ to $eq(\mathbf{t_1}, \mathbf{t_2})$, the resulting Datalog program can handle equalities of the individuals names.

This modification will not change the complexity of the resulting Datalog program in term of complexity theory. However it may affect the performance of the datalog engines, because they can not use the native identity checking of the constants.

4.6 Discussion

In this section we discuss some aspects of inline evaluation of dl-programs.

4.6.1 Direct Rewriting vs Reification based Rewriting

We can classify the Datalog rewritings of DLs into two styles based on how to handle the signature of the ontology. The first style is called *direct rewriting*, which is used in $\mathcal{LDL}^+$ and Horn-$\mathcal{SHIQ}$, where the resulting Datalog programs use predicates from concept- and role names and convert the ABox assertions directly into facts. Another style is called *reification based rewriting* and used in $\mathcal{EL}$. Recall that the $\mathcal{EL}$ rewriting uses fixed set of predicates and inference rules, and all the TBox and ABox assertions are reified to Datalog facts. In the OWL 2 specification [Mot+08] of W3C, the algorithm for OWL 2 RL is also reification based.

Non-reified rewritings normally preserve the structure of the ontologies, e.g. acyclicity, and Datalog reasoners can take the advantage of such properties to optimize the evaluation. Reified rewritings normally use fixed predicates, so they are more suitable to store the facts in RDBMS. However, reified program is usually recursive, and the original structure of the ontology is often lost, so it might be more difficult to evaluate on Datalog engine.

4.6.2 Datalog Reasoner vs Proprietary Reasoner

Our Datalog rewriting is strongly related to the proprietary rule based consequence driven reasoning [Kaz09b; Krö11; SKH11] for DLs. This technique has been successfully used in $\mathcal{EL}$ [Krö11], Horn-$\mathcal{SHIQ}$ [Kaz09b] and $\mathcal{ALCH}$ [SKH11] and is implemented in the systems ELK[5], CB[6], and ConDOR[7]. They have a fixed set of inference rules (not necessarily in Datalog format), and implement the (optimized) evaluation of these rules directly in some programming languages (e.g., Java for ELK, Ocaml for CB, and C for ConDOR).

The advantage of these proprietary engines is that they can fully control the evaluation, and implement their dedicated optimizations. However, such systems normally need more effort to develop. In case of Datalog reasoning in our framework, we can reuse the existing Datalog engines; the systems are more flexible since changing the inference rules is much easier.

4.6.3 Relations to other Datalog Rewritings

Reductions of DLs to LP have been considered before, e.g., in [HMS04; Swi04]. Swift reduces reasoning in the DL $\mathcal{ALCQI}$ (in fact, consistency checking of concept expressions) to Datalog$^{\neg}$ under answer set semantics (employing a guess and check methodology) [Swi04], while Hustadt *el al.* reduce reasoning in the DL $\mathcal{SHIQ}^{-}$ to disjunctive Datalog by resolution [HMS04] and the technique is implemented in KAON2.

With some trivial modification, the rewriting results in KAON2 could be possibly used in our framework of inline evaluation. It is worth noting that [SKH11] points out although theoretically optimal, compared with consequence-based approaches, resolution-based procedures normally generate too many redundant intermediate clauses.

4.6.4 Non-Horn Description Logics

All the Datalog-rewritable description logics we considered so far ($\mathcal{LDL}^{+}$, $\mathcal{EL}$, Horn-$\mathcal{SHIQ}$) are Horn logics, where we can not freely use disjunctions. Most of the non-Horn Logics ($\mathcal{ELU}$, $\mathcal{ALC}$, $\mathcal{SHIQ}$) are coNP complete under data complexity [KL07; HMS07]. Recall that Datalog is P complete under data complexity. Therefore non-Horn Descriptions cannot be reduced to Datalog, unless P = coNP. Furthermore, recently Grau *el al.* proved a stronger result that it is impossible in general to answer queries over non-Horn ontologies using Datalog rewritings even for very simple ontology languages, and even if P = NP [Gra+13]

The data complexity of Datalog$^{\vee}$ is also coNP complete. To handle disjunction, instead of Datalog in non-Horn DL reasoning, we may need Datalog$^{\vee}$. For example, KAON2 system reduces the $\mathcal{SHIQ}$ ontologies to Datalog$^{\vee}$ for instance query.

However, we should be careful that we can not simply plugin the Datalog$^{\vee}$ program produced by KAON2 into our inline evaluation framework. The reason is that Datalog$^{\vee}$ is an inference based system doing cautious reasoning; while in dl-programs we need a model-based system. Actually in dl-programs disjunctive information from a DL knowledge base is lost to

[5] https://code.google.com/p/elk-reasoner/
[6] https://code.google.com/p/cb-reasoner/
[7] https://code.google.com/p/condor-reasoner/

the ASP program [VDB10]. For instance, consider a dl-program $\mathcal{KB} = (L, P)$, where

$$L = \{Person \sqsubseteq Woman \sqcup Man, Person(Alex)\},$$

and P is the following program:

$$P(x) \leftarrow DL[Woman](x).$$
$$P(x) \leftarrow DL[Man](x).$$

Then $\mathcal{KB}$ does not actually imply $P(Alex)$, because the DL knowledge base cannot derive either $Woman(Alex)$ or $Man(Alex)$. If we directly replace the DL axiom $Person \sqsubseteq Woman \sqcup Man$ by the KAON2 rewriting result

$$Woman(x) \vee Man(x) \leftarrow Person(x)$$

and use the inline framework rewriting, we will get two answer sets, and $P(Alex)$ is in both answer sets. However, this is different from the semantics of dl-program.

4.6.5 Operators $\uplus$ and $\curlywedge$

We have not considered the operators $\uplus$ and $\curlywedge$ in this chapter yet.

The operator $\uplus$ can be easily removed by simply replacing all the input $A \uplus p$ with $(\neg A) \uplus p$. Then we need to carefully handle the negative assertions. See section 5.1 for how to deal with negative assertions in $\mathcal{EL}$ ontologies.

The operator $\curlywedge$ is more difficult to handle. Wang *el al.* recently developed a technique eliminating the nonmonotonic dl-atoms (using $\curlywedge$) [Wan+13]. This technique can help us handling some operators $\curlywedge$ and the step of eliminating is not difficult. However, deciding whether a dl-atom is monotonic is in general intractable.

4.6.6 Relaxation of Datalog-rewritability

We can consider here relaxed notions of Datalog-rewritability that allow for auxiliary relations. For Datalog$^\neg$ rewritability of a dl-program, we required that dl-atoms resp. the underlying description logic are Datalog rewritable. One also could consider a more liberal notion of Datalog$^\neg$ rewritability of dl-atoms, which however would need to deal with the fact that in principle, the program $\Phi_{\mathcal{DL}}(L)$ could have multiple answer sets (or none). Furthermore, simply plugging in $\Phi_{\mathcal{DL}}(L_\lambda)$ for some dl-atom $DL[\lambda, Q](\vec{t}$ may lead to unwanted effects, due to nonmonotonicity; e.g., additional answer sets might emerge. Syntactic restrictions (e.g., acyclicity, or more general that dl-atoms are not involved in cycles) can avoid this.

We considered some formalisms for uniform evaluation of dl-programs, For which evaluation engines are available. Of course, further such formalisms (e.g., FO(·) Logic [DT08], or F-logic [KLW95]) may be considered. But also formalisms for which reasoning engines are yet emerging might be of interest, e.g. Datalog $^\pm$ [Cal+10]. The latter extends Datalog with existential quantification in rule head and at the same time restricts the syntax such that reasoning remains decidable. Datalog $^\pm$ is more expressive than various description logics, including

DL-Lite (which is captured elegantly), and allows for handling unknown individuals in the reasoning. It seems to be an attractive formalism in particular to host the combination of rules and ontologies.

CHAPTER 5

Inline Evaluation of other Hybrid Knowledge Bases

In this chapter, we show that the ideas of inline evaluation can be used in other formalisms of hybrid knowledge bases.

- We start with terminological default logic, which can be rewritten to dl-programs. The resulting programs involve inferencing of negative queries; we extend Datalog rewriting of $\mathcal{EL}$ to handle negative queries.
- Next, we show that DL-safe conjunctive queries can be rewritten to dl-programs naturally.
- Then we move to the general conjunctive queries and weakly dl-safe KBs over $\mathcal{SHIQ}$ ontologies. We develop a query rewriting algorithm reducing these reasoning tasks to evaluating Datalog programs.
- Finally, by combining the techniques of inline evaluation for dl-programs and query rewriting, we extend the inline evaluation framework to cq-programs.

5.1 Terminological Default Logics

Default Logics

Default logic is a well-established nonmonotonic reasoning system proposed by Reiter [Rei80]. A default theory $\Delta = (W, D)$ consists of a set W of first-order sentences and a set D of defaults of the form

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \tag{5.1}$$

where α, all β_i, and γ are (possibly open) first-order formulas. Reiter defines the semantics of default logic by the extensions as the fixpoints of the operator $\Gamma_\Delta(S)$ as follows. For a set of sentences S, $\Gamma_\Delta(S)$ is the least set of sentences such that

(1) $W \subseteq \Gamma_\Delta(S)$;

(2) $Cn(\Gamma_\Delta(S)) = \Gamma_\Delta(S)$;

(3) if $\frac{\alpha:\beta_1,\ldots,\beta_n}{\gamma} \in D, \alpha \in \Gamma_\Delta(S)$, and $\neg\beta_1, \ldots, \neg\beta_n \in S$ then $\gamma \in \Gamma_\Delta(S)$.

Then, a set of formulas E is an extension of Δ iff $E = \Gamma_\Delta(E)$. Extensions of open default theories (i.e., default theories which are not closed) are defined via ground instances of defaults. In case the defaults do not contain quantifiers, the grounding is defined analogously to logic program rules; otherwise, skolemization step is required (see [Rei80] for details).

Terminological Default Logics

A terminological default is a default theory $\Delta = (L, D)$, where L is a DL knowledge base, and D consists of certain quantifier-free defaults [BH95]. We show this here for defaults of form (5.1) where α is a conjunction of literals and all β_i's and γ are literals. Here, Reiter-style default rules are applied to named individuals for decidability. We show the classic birds&penguins example for illustration.

Example 5.1. *The terminology default knowledge base $\Delta = \langle L, D\rangle$ consists of an $\mathcal{EL}$ ontology $L = \{Flier \sqcap NonFlier \sqsubseteq \bot, Penguin \sqsubseteq Bird,\ Penguin \sqsubseteq NonFlier,\ Bird(tweety)\}$, and a (singleton) set $D = \{Bird(X) : Flier(X)/Flier(X)\}$ of default rules (informally, birds fly by default).*

Transformation of Terminological Defaults to DL-Programs

There are several transformations of terminological defaults to dl-programs [Eit+08a; DEK09]. We revisit the transformation Π.

Definition 5.2. *For each default δ of form*

$$\frac{\alpha(\mathbf{X}) : \beta_1(\mathbf{Y_1}), \ldots, \beta_m(\mathbf{Y_m})}{\gamma(\mathbf{Z})}$$

(where the β_i and γ are just literals), the transformation $\Pi(\delta)$ uses the following dl-rules:

$$in\_\gamma(\mathbf{Z}) \leftarrow not\ out\_\gamma(\mathbf{Z}); \quad out\_\gamma(\mathbf{Z}) \leftarrow not\ in\_\gamma(\mathbf{Z}) \tag{5.2}$$

$$g(\mathbf{Z}) \leftarrow DL[\lambda;\alpha_1](\mathbf{X_1}), \ldots, DL[\lambda;\alpha_k](\mathbf{X_k}), \quad not\ DL[\lambda';\neg\beta_1](\mathbf{Y_1}), \ldots, not\ DL[\lambda';\neg\beta_m](\mathbf{Y_m}) \tag{5.3}$$

$$fail \leftarrow DL[\lambda';\gamma](\mathbf{Z}), out\_\gamma(\mathbf{Z}), not\ fail \tag{5.4}$$

$$fail \leftarrow not\ DL[\lambda;\gamma](\mathbf{Z}), in\_\gamma(\mathbf{Z}), not\ fail \tag{5.5}$$

$$fail \leftarrow DL[\lambda;\gamma](\mathbf{Z}), out\_\gamma(\mathbf{Z}), not\ fail \tag{5.6}$$

where λ' contains for each default δ an update $\gamma^* \uplus in\_\gamma$ if $\gamma(\mathbf{Z})$ is positive, and an update $\gamma^* \uplus in\_\gamma$ if $\gamma(\mathbf{Z})$ is negative; λ is similar with g in place of in_γ.

The transformation Π can be naturally to a set of defaults D as $\Pi(D) = \bigcup_{\delta\in D} \Pi(\delta)$.

Intuitively, the transformation Π implements a guess-and-check strategy, by which a sufficiently large part of an extension E, including all conclusions γ of applied defaults δ, is guessed using predicates in_γ and out_γ and described by an update λ' of L, in which we have $p \uplus in\_\gamma$, where p is the predicate of γ , if the literal γ is positive and $p \uplus in\_\gamma$ otherwise. The candidate E is then checked using a predicate g for γ to characterize $\Gamma_\Delta(E)$, which is described by an update λ of L which includes $p \uplus g$ if γ is a positive literal and $p \uplus g$ otherwise.

Reasoning over Terminological Defaults on $\mathcal{EL}$ ontologies

The resulting dl-programs of Π involves query of negative concepts (e.g. in $DL[\lambda'; \neg\beta_1](\mathbf{Y_1})$). Recall that techniques we developed in Section 4.5 only works for ground queries. Here we present an elegant Datalog transformation answering instance queries with variables over $\mathcal{EL}$ ontologies.

Trivially, $L \models \neg C(a)$ is equivalent to unsatisfiability of $L \cup \{C(a)\}$. To answer a negative query $\neg C(X)$, we need to bind X to every possible individual, and reduce it to unsatisfiability checking. This one by one checking can be elegantly achieved via datalog encoding. The idea is to extend $isa/2$ with two more arguments, representing the individual and the concept name, to $isa\_n/4$; in P_{inst}, each $isa(X, Y)$ is uniformly replaced with $isa\_n(X, Y, C, J)$, and each $triple(X, Y, Z)$ uniformly with $triple\_n(X, Y, Z, C, J)$, yielding $P^{\neg}_{inst}$. This set includes e.g. the following rules, which propagate subclass and conjunctive subclass membership:

$$
\begin{array}{ll}
isa\_n(X, Z, C, J) & \leftarrow subClass(Y, Z), isa\_n(X, Y, C, J) \\
isa\_n(X, Z, C, J) & \leftarrow subConj(Y_1, Y_2, Z), isa\_n(X, Y_1, C, J), isa\_n(X, Y_2, C, J) \\
isa\_n(X, Z, C, J) & \leftarrow subEx(V, Y, Z), triple\_n(X, V, X', C, J), isa\_n(X', Y, C, J)
\end{array}
$$

The individual unsatisfiability checks are then accomplished with rules $P^{\neg}$, which for each check add $C(a)$, expand all $isa(X, Y)$ atoms with a and C, and make an atom $isnota(a, C)$ true iff the test is successful:

$$isa\_n(X, Y, Y, X) \leftarrow nom(X), cls(Y) \tag{5.7}$$

$$isa\_n(X_1, Y_1, Y_2, X_2) \leftarrow isa(X_1, Y_1), cls(Y_2), nom(X_2) \tag{5.8}$$

$$isnota(X, Y) \leftarrow isa\_n(N, Z, Y, X), nom(N), bot(Z), nom(X) \tag{5.9}$$

Let the extended $\mathcal{EL}$ reduction for negative query of an $\mathcal{EL}$ ontology L be defined as

$$\Phi^{\neg}_{\mathcal{EL}}(L) = P^{\neg}_{inst} \cup \{I_{inst}(\alpha) \mid \alpha \in L\} \cup \{I_{inst}(s) \mid s \in N_I \cup N_C \cup N_R\} \cup P^{\neg}.$$

Proposition 5.3. *For every $\mathcal{EL}$ ontology L, concept C, and individual a, we have $L \models \neg C(a)$ iff $\Phi^{\neg}_{\mathcal{EL}}(L) \models isnota(a, C)$.*

Proof. By comparing the structure of $\Phi_{\mathcal{EL}}$ and $\Phi^{\neg}_{\mathcal{EL}}$, we can show that $\Phi_{\mathcal{EL}}(L \cup \{C(a)\}) \models$

$isa(x, y)$ if and only if $\Phi^{\neg}_{\mathcal{EL}} \models isa\_n(x, y, C, a)$ for any constants x, y. Then

$L \models \neg C(a)$
iff $L \cup \{C(a)\}$ is unsatisfiable
iff $L \cup \{C(a)\} \models \bot(n)$ for some individual n
iff $\Phi_{\mathcal{EL}}(L \cup \{C(a)\}) \models isa(n, z)$ and $\Phi_{\mathcal{EL}}(L \cup \{C(a)\}) \models nom(n)$
and $\Phi_{\mathcal{EL}}(L \cup \{C(a)\}) \models bot(z)$
iff $\Phi^{\neg}_{\mathcal{EL}}(L) \models isa\_n(n, z, C, a)$ and $\Phi^{\neg}_{\mathcal{EL}}(L) \models nom(n)$ and $\Phi^{\neg}_{\mathcal{EL}}(L) \models bot(z)$
[by Rule 5.9]
iff $\Phi^{\neg}_{\mathcal{EL}}(L) \models isnota(a, C)$

□

Example 5.4. *Consider the terminological default KB in Example 5.1. The semantics of the KB Δ is captured by the following dl-program $(L, \Pi(D))$ under answer set semantics (i.e., default extensions correspond to answer sets), where $\Pi(D)$ is*

$$in\_Flier(X) \leftarrow \textit{ not } out\_Flier(X) \tag{5.10}$$
$$out\_Flier(X) \leftarrow \textit{ not } in\_Flier(X) \tag{5.11}$$
$$Flier^{+}(X) \leftarrow DL[\lambda; Bird](X),\ \textit{not } DL[\lambda'; \neg Flier](X) \tag{5.12}$$
$$fail \leftarrow DL[\lambda'; Flier](X), out\_Flier(X),\ \textit{not fail} \tag{5.13}$$
$$fail \leftarrow DL[\lambda; Flier](X), in\_Flier(X),\ \textit{not fail} \tag{5.14}$$
$$fail \leftarrow DL[\lambda; Flier](X), out\_Flier(X),\ \textit{not fail} \tag{5.15}$$

where $\lambda = \{Flier \uplus in\_Flier\}$ and $\lambda' = \{Flier \uplus Flier^{+}\}$.

To rewrite $\mathcal{KB}$ to Datalog$^{\neg}$*, we replace all the dl-atoms $DL[\lambda; Q]$ by Q_λ and $DL[\lambda; \neg Q]$ by $(\neg Q)_\lambda$. For example, rule (5.12) is replaced by*

$$Fly^{+}(x) \leftarrow Bird_\lambda(X), not(\neg Fly)_{\lambda'}(X).$$

The rules for defining $Bird_\lambda$ and $(\neg Fly)_{\lambda'}$ are:

$$Bird_\lambda(X) \leftarrow isa_\lambda(X, Bird)$$
$$(\neg Fly)_{\lambda'}(X) \leftarrow isnota_{\lambda'}(X, Fly)$$

We skip the rest of the rewriting, as the are straightforward from the inline evaluation framework. The Datalog$^{\neg}$ *program has a single answer set, which contains $Flier(tweety)$, as expected.*

Reasoning over Terminological Defaults on other DLs

The technique used for answering negative concept queries $C(X)$ in $\mathcal{EL}$ can also be used for the other DL, e.g., $\mathcal{LDL}^{+}$ and Horn-$\mathcal{SHIQ}$ by extending the predicates in the rewritten Datalog with additional two arities for C and X. Then reasoning of terminological Defaults over other DLs can be handled similarly in the inline evaluation framework.

5.2 DL-safe Conjunctive Queries

In this section, we consider DL-safe conjunctive queries (CQs) (i.e. CQs interpreted under DL-safeness condition; see 3.18). Recall that, given a DL vocabulary $(\mathsf{N_C}, \mathsf{N_R})$, a CQ ρ can be represented as a rule of the form:

$$ans(\mathbf{X}) \leftarrow s_1(\mathbf{Y_1}), \ldots, s_n(\mathbf{Y_n}) \tag{5.16}$$

where $s_i \in \mathsf{N_C} \cup \mathsf{N_R}$ and ans is a fresh predicate.

DL-safeness requires that every variable in the rule must also appear in some positive non-DL-atoms of the body (Definition 3.17). Clearly, rule (5.16) is not DL-safe in general, because there is no non-DL predicate in the rule body. To make it DL-safe, one can (1) append auxiliary atoms $\mathcal{O}(X)$ to the body for all the variables inside it and (2) add facts $\mathcal{O}(a)$ for all the named individual a to the program.

Equivalently, we can transform CQ (5.16) into a dl-rule by replacing every atom $s_i(\mathbf{Y_i})$ with the dl-atom $\mathrm{DL}[s_i](\mathbf{Y_i})$ having empty input list:

$$ans(\mathbf{X}) \leftarrow DL[s_1](\mathbf{Y_1}), \ldots, DL[s_n](\mathbf{Y_n}). \tag{5.17}$$

The equivalence is guaranteed by the semantics of the dl-atoms in dl-programs, where only named individuals can be results of the dl-query. When the DL component is Datalog-writable, we can apply the inline evaluation framework to the resulting dl-program.

Example 5.5. *The following CQ retrieves pairs of wired $HighTrafficNode$ X and Y:*

$$ans(X, Y) \leftarrow HighTrafficNode(X), wired(X, Y), HighTrafficNode(Y)$$

It can be converted to a DL-rule:

$$ans(X, Y) \leftarrow \mathbf{DL}[HighTrafficNode](X), \mathbf{DL}[wired](X, Y), \mathbf{DL}[HighTrafficNode](Y)$$

5.3 Conjunctive Queries and Positive Weakly DL-safe KBs

In this section, we show how to answer conjunctive queries and positive weakly DL-safe KBs over Horn-$\mathcal{SHIQ}$ ontologies. Based on the results of section 4.4, we develop a writing algorithm for CQ over Horn-$\mathcal{SHIQ}$, and we show that this algorithm can be naturally extended to positive weakly DL-safe KBs.

The following is immediate from Propositions 4.33 and 4.37:

Theorem 5.6. *Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be a Horn-$\mathcal{SHIQ}$ ontology and q be a CQ. Then $\mathcal{A} \cup \mathsf{cr}(\mathcal{T}^*)$ is consistent iff $\mathcal{O}$ is consistent. Moreover, if $\mathcal{O}$ is consistent, then $ans(\mathcal{O}, q) = ans^{\mathcal{T}}(\mathcal{I}_{\mathcal{O}}, q)$ for every CQ q, where $\mathcal{I}_{\mathcal{O}} = chase(MM(\mathcal{A} \cup \mathsf{cr}(\mathcal{T}^*), \Xi(\mathcal{T}^*))$.*

Computing $ans^{\mathcal{T}}(\mathcal{I}_{\mathcal{O}}, q)$ is still tricky because $\mathcal{I}_{\mathcal{O}}$ can be infinite. Hence we rewrite q into a set Q of CQs such that $ans^{\mathcal{T}}(\mathcal{I}_{\mathcal{O}}, q) = \bigcup_{q' \in Q} ans(MM(\mathcal{A} \cup \mathsf{cr}(\mathcal{T}^*), q')$. That is, we only need to evaluate the queries in Q over the Datalog program $\mathcal{A} \cup \mathsf{cr}(\mathcal{T}^*)$. Since this can be easily done directly in Datalog, we have an algorithm for answering q over $\mathcal{O}$, which we later generalize to positive weakly DL-safe KBs.

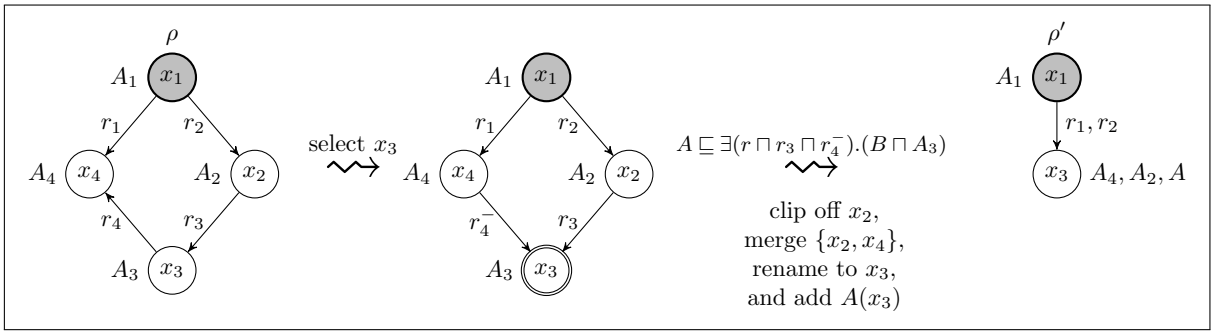


Figure 5.1: Example 5.7, query rewriting with only simple roles

5.3.1 Rewriting rules with simple roles only

We will first present a simplified version of our rewriting algorithm that rewrites a rule ρ assuming that r is a simple role for all atoms of the form $r(x, y)$ that occur in its body. This version can be explained more easily, and it will allow us to give a better explanation of the general algorithm.

The intuition is the following. Suppose that ρ has a non-distinguished variable x, and that there is some match π in $\mathcal{I}_\mathcal{O}$ such that $\pi(x)$ is an object in the 'tree part' introduced by the chase procedure and it has no descendant in the image of π, that is, $\pi(x)$ it is a leaf in the forest shaped image of ρ under π. Then for all atoms $r(y, x)$ of ρ, the "neighbor" variable y must mapped to the parent of $\pi(x)$. A rewrite step makes a choice of such an x, and employs an existential axiom from $\Xi(\mathcal{T})$ to 'clip off' x, eliminating all query atoms that mention it. By repeating this procedure, we can clip off all variables matched in the tree part and obtain a rule that has a match in $MM(\mathcal{A} \cup \mathsf{cr}(\mathcal{T}))$.

The one-step clipping off works as follows. For a CQ ρ and a Horn-$\mathcal{ALCHIQ}^\sqcap$ TBox $\mathcal{T}$, we write $\rho \to_\mathcal{T} \rho'$ if ρ' can be obtained from ρ in the following steps:

(S1) Select in ρ an arbitrary non-distinguished variable x such that there are no atoms of the form $r(x, x)$ in ρ.

(S2) Replace each role atom $r(x, y)$ in ρ, where y is arbitrary, by the atom $\mathsf{inv}(r)(y, x)$.

(S3) Let $V_p = \{y \mid \exists r : r(y, x) \in \mathit{body}(\rho)\}$, and select some $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T})$ such that

  (a) $\{r \mid r(y, x) \in \mathit{body}(\rho) \wedge y \in V_p\} \subseteq S$, and

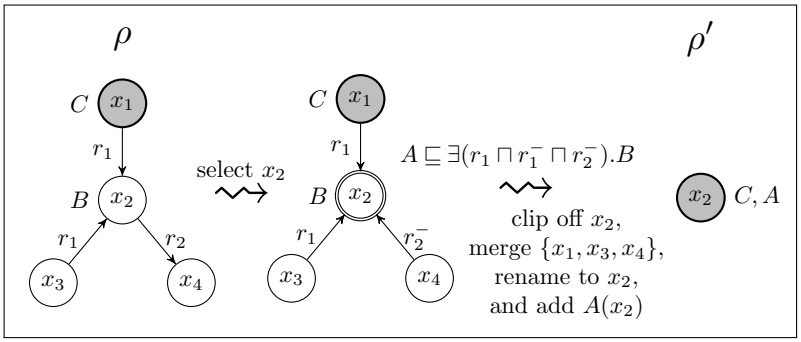
  (b) $\{A \mid A(x) \in \mathit{body}(\rho)\} \subseteq N$.

(S4) Drop from ρ each atom containing x.

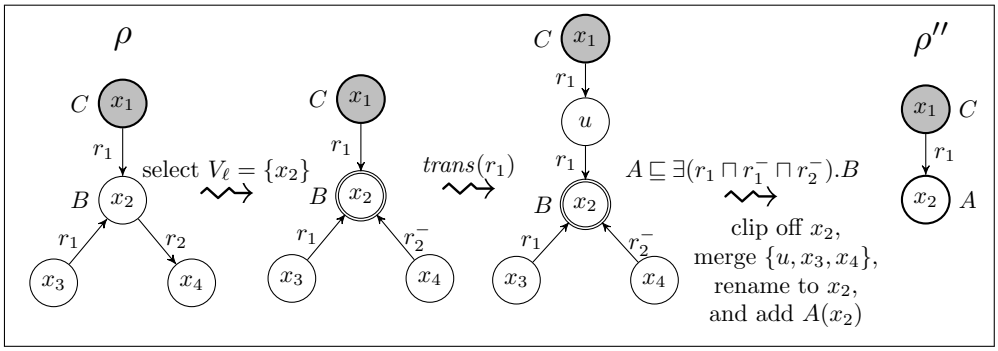
(S5) Rename each $y \in V_p$ of ρ by x.

(S6) Add the atoms $\{A(x) \mid A \in M\}$ to $\mathit{body}(\rho)$.

We illustrate the rewriting step with two examples:



(a) Example 5.8: Query rewriting with only simple roles



(b) Example 5.10: Query rewriting with non-simple roles

Example 5.7. *Let* $\rho : q(x_1) \leftarrow A_1(x_1), r_2(x_1, x_2), A_2(x_2), r_3(x_2, x_3), A_3(x_3), r_1(x_1, x_4), A_4(x_4), r_4(x_3, x_4)$ *in Figure 5.1, and assume that* $A \sqsubseteq \exists(r \sqcap r_3 \sqcap r_4^-).(B \sqcap A_3)$ *is in* $\Xi(\mathcal{T})$ *and that all roles are simple. We choose the variable* x_3*, replace* $r_4(x_3, x_4)$ *by* $r_4^-(x_4, x_3)$ *in step (S2), and get* $V_p = \{x_2, x_4\}$*. Intuitively, if* $\pi(x_3)$ *is a leaf in a tree-shaped match* π*, then* x_2 *and* x_4 *must both be mapped to the parent of* $\pi(x_3)$*. Since the GCI* $A \sqsubseteq \exists(r \sqcap r_3 \sqcap r_4^-).(B \sqcap A_3)$ *in* $\Xi(\mathcal{T})$ *satisfies (S3.a,b), we can drop the atoms containing* x_3 *from* ρ*, and perform (S5) and (S6) to obtain the rewritten query* $\rho' : q(x_1) \leftarrow A_1(x_1), r_1(x_1, x_3), r_2(x_1, x_3), A_4(x_3), A_2(x_3), A(x_3)$*.*

Example 5.8. *In this example, illustrated in Figure 5.2a, we again assume that all roles are simple. Let* $\rho : q(x_1) \leftarrow C(x_1),\ B(x_2),\ r_1(x_1, x_2),\ r_1(x_3, x_2),\ r_2(x_2, x_4)$*, and assume* $A \sqsubseteq \exists(r_1 \sqcap r_1^- \sqcap r_2^-).B \in \Xi(\mathcal{T}^*)$*. In (S1) we select the non-distinguished variable* x_2*. Next, in (S2), we replace* $r_2(x_2, x_4)$ *by* $r_2^-(x_4, x_2)$*. Since all roles are simple, we do nothing in (S3). In (S4) we choose* $V_\ell = \{x_2\}$ *and* $V_p = \{x_1, x_3, x_4\}$*, and in (S5),* $A \sqsubseteq \exists(r_1 \sqcap r_1^- \sqcap r_2^-).B$*. Then we clip off* x_2 *in (S6), merge all variables in* V_p *and rename them to* x_2 *in (S7), and add* $A(x_2)$ *in (S8), to obtain* $\rho' : q(x_2) \leftarrow C(x_2), A(x_2)$*.*

5.3.2 Rewriting arbitrary rules

Now we present the rewriting algorithm for the general case, and show that it is sound and complete.

As above, suppose that ρ has a match and $\pi(x)$ is a leaf of its forest shaped image, for some variable x. The most significant difference in the presence of non-simple roles is that if the query has an atom $r(y, x)$ and r is non-simple, then $\pi(y)$ is not necessarily the parent p of $\pi(x)$. Instead, $\pi(y)$ can be an ancestor of p, or $\pi(y) = \pi(x)$ may hold. Hence, instead of just x, we guess a set of distinguished variables V_ℓ that are mapped together at some leaf node $\pi(x)$. Then we guess a subset of the neighbor variables whose match is higher up in the tree, and for them we introduce an 'intermediate' variable u that can be matched at the parent p. In this way we can forget about the variables that are matched to ancestors of p, and assume that all the neighbours V_p of the variables in V_ℓ are matched at p. We can then proceed similarly as above and clip off all variables in V_ℓ using an axiom from $M \sqsubseteq \exists S.N$ that ensures the existence of a match for them. This axiom must now also ensure that $\pi(x)$ is an r-successor of itself for every atom $r(x, y)$ such that $x, y \in V_\ell$. This is verified by the new condition (S5c), which relies on the fact that a node e is an r-successor of itself in $\mathcal{I}_\mathcal{O}$ iff both $e, e' \in s^{\mathcal{I}_\mathcal{O}}$ and $e', e \in s^{\mathcal{I}_\mathcal{O}}$ hold for some transitive $s \sqsubseteq^*_\mathcal{T} r$, where e' is either the parent or a child of e in $\mathcal{I}_\mathcal{O}$.

Definition 5.9. *For a rule ρ and a Horn-$\mathcal{SHIQ}$ TBox $\mathcal{T}$, we write $\rho \to_\mathcal{T} \rho'$ if ρ' is obtained from ρ by the following steps:*

(S1) Select an arbitrary non-empty set V_ℓ of non-distinguished variables in ρ.

(S2) Replace each role atom $r(x, y)$ in ρ, where $x \in V_\ell$ and $y \notin V_\ell$ is arbitrary, by the atom $\mathsf{inv}(r)(y, x)$.

(S3) For each atom $\alpha = s(y, x)$ in ρ, where where $x \in V_\ell$, $y \notin V_\ell$ is arbitrary and s is non-simple, either leave α untouched or replace it by two atoms $r(y, u), r(u, x)$, where u is a fresh variable and r is a transitive role with $r \sqsubseteq^*_\mathcal{T} s$.

(S4) Let $V_p = \{y \mid \exists r : r(y, x) \in body(\rho) \wedge x \in V_\ell \wedge y \notin V_\ell\}$.

(S5) Select some $M \sqsubseteq \exists S.N \in \Xi(\mathcal{T}^*)$ such that

- *(a) $\{r \mid r(y, x) \in body(\rho) \wedge x \in V_\ell \wedge y \in V_p\} \subseteq S$,*
- *(b) $\{A \mid A(x) \in body(\rho) \wedge x \in V_\ell\} \subseteq N$, and*
- *(c) for each atom $r(x, y)$ in $body(\rho)$ with $x, y \in V_\ell$ there is a transitive $s \sqsubseteq^*_\mathcal{T} r$ such that*
  - *i. $\{s, s^-\} \subseteq S$, or*
  - *ii. there is an axiom $M' \sqsubseteq \exists S'.N' \in \Xi(\mathcal{T}^*)$ such that $M' \subseteq N$ and $\{s, s^-\} \subseteq S'$.*

(S6) Drop each atom from ρ containing a variable from V_ℓ.

(S7) Select some $x \in V_\ell$ and rename each $y \in V_p$ of ρ by x.

(S8) Add the atoms $\{A(x) \mid A \in M\}$ to ρ.

We write $\rho \to^*_\mathcal{T} \rho'$ if ρ' can be obtained from ρ by finitely many rewrite iterations. We let $\mathsf{rew}_\mathcal{T}(\rho) = \{\rho' \mid \rho \to^*_\mathcal{T} \rho'\}$. For a set $\mathcal{P}$ of rules, $\mathsf{rew}_\mathcal{T}(\mathcal{P}) = \bigcup_{\rho \in \mathcal{P}} \mathsf{rew}_\mathcal{T}(\rho)$.

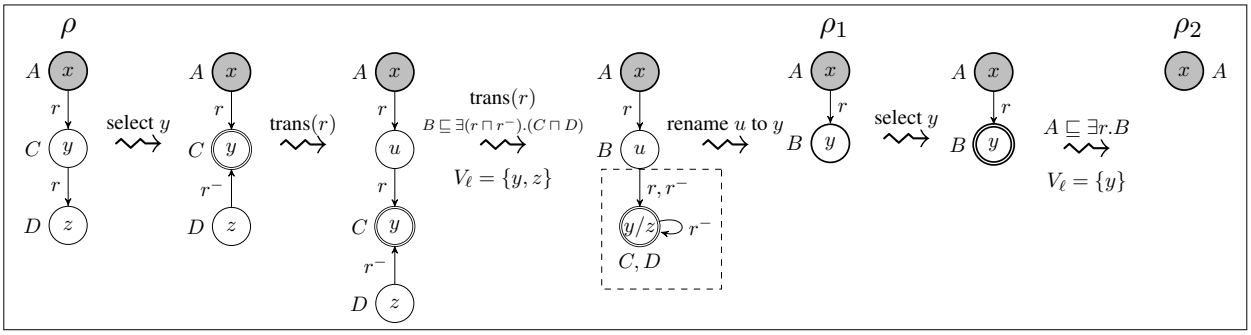


Figure 5.3: Example 5.11

Example 5.10 (ctd)**.** *Recall $\rho : q(x_1) \leftarrow C(x_1),\ B(x_2),\ r_1(x_1, x_2),\ r_1(x_3, x_2),\ r_2(x_2, x_4)$ and $A \sqsubseteq \exists(r_1 \sqcap r_1^- \sqcap r_2^-).B \in \Xi(\mathcal{T}^*)$ from Example 5.8, but now assume that $trans(r_1) \in \mathcal{T}$. As shown in Figure 5.2b, in (S1) we choose $V_\ell = \{x_2\}$, and in (S3) we choose to replace $r_1(x_1, x_2)$ with $r_1(x_1, u)$, $r_1(u, x_2)$. In (S4) we get $V_p = \{u, x_3, x_4\}$. Then we proceed similarly as above to obtain $\rho'' : q(x_1) \leftarrow C(x_1), r_1(x_1, x_2), A(x_2)$.*

Example 5.11. *Assume $\mathcal{T} = \{r \sqsubseteq r^-,\ trans(r),\ A \sqsubseteq \exists r.B,\ B \sqsubseteq \exists r.C,\ C \sqsubseteq D\}$. Let $\rho : q(X) \leftarrow A(x), r(x, y), C(y), D(z), r(y, z)$. By saturation rules $\mathbf{R}^c_\sqsubseteq$ and $\mathbf{R}^r_\sqsubseteq$, we have $B \sqsubseteq \exists(r \sqcap r^-).(C \sqcap D) \in \Xi(\mathcal{T})$.*

In the first round, in (S1) we select y. In (S2), we replace $r(y, z)$ by $r^-(z, y)$. In (S3), as r is transitive, we replace $r(x, z)$ by $r(x, u)$ and $r(u, y)$. In (S4), we choose $V_\ell = \{y, z\}$, $V_p = \{u\}$. In (S5), we choose $B \sqsubseteq \exists(r \sqcap r^-).(C \sqcap D) \in \Xi(\mathcal{T})$, which satisfies (S5.a), (S5.b), and (S5.c1). In (S6), we drop atoms containing y or z from $body(\rho)$. In (S7), we rename u to y. Finally in (S8), we add $B(y)$ to the body and get $\rho_1 : q(x) \leftarrow A(x), r(x, y), B(y)$.

In the second round, we select y in (S1), $V_\ell = \{y\}, V_p = \{x\}$ in (S3), and $A \sqsubseteq \exists r.B$ in (S5). Following the similar steps, we get another rewritten rule $\rho_2 : q(X) \leftarrow A(x)$.

The following is crucial:

Theorem 5.12. *Assume a consistent Horn-$\mathcal{SHIQ}$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ and a conjunctive query q. Then $ans(\mathcal{O}, q) = \bigcup_{q' \in \mathsf{rew}_\mathcal{T}(q)} ans(MM(\mathcal{A} \cup \mathsf{cr}(\mathcal{T}^*)), q')$.*

Proof. Let $\mathcal{I}_\mathcal{O} = chase(\mathcal{J}, \Xi(\mathcal{T}^*))$, where $\mathcal{J} = MM(\mathcal{A} \cup \mathsf{cr}(\mathcal{T}^*))$. It suffices to show $ans^\mathcal{T}(\mathcal{I}_\mathcal{O}, q) = ans(\mathcal{J}, \mathsf{rew}_\mathcal{T}(q))$.

We first show $ans^\mathcal{T}(\mathcal{I}_\mathcal{O}, q) \supseteq ans(\mathcal{J}, \mathsf{rew}_\mathcal{T}(q))$. Suppose $h(\vec{x})$ is the head atom of q. Assume a tuple $\vec{u} \in ans(\mathcal{J}, \mathsf{rew}_\mathcal{T}(q))$. Then there is a query $q' \in \mathsf{rew}_\mathcal{T}(q)$ and a match $\pi_{q'}$ for q' in $\mathcal{J}$ such that $\vec{u} = \pi_{q'}(\vec{x})$. By the construction of $\mathcal{I}_\mathcal{O}$, we also have $\vec{u} \in ans^\mathcal{T}(\mathcal{I}_\mathcal{O}, q')$. If $q' = q$, then we are done. Suppose $q' \neq q$. Then there is $n > 0$ such that $q_0 \rightarrow_\mathcal{T} q_1, \cdots, q_{n-1} \rightarrow_\mathcal{T} q_n$ with $q_0 = q$ and $q_n = q'$. Thus to prove the claim it suffices to show that $\vec{u} \in ans^\mathcal{T}(\mathcal{I}_\mathcal{O}, q_i)$ implies $\vec{u} \in ans^\mathcal{T}(\mathcal{I}_\mathcal{O}, q_{i-1})$, where $0 < i \leq n$.

Suppose π_{q_i} is a match for q_i in $\mathcal{I}_\mathcal{O}$ with $\vec{u} = \pi_{q'}(\vec{x})$, i.e. $\vec{u} \in ans^\mathcal{T}(\mathcal{I}_\mathcal{O}, q_i)$. Let V_ℓ be the set chosen in (S1), let $x \in V_\ell$ be the variable chosen in (S7), and let $M \sqsubseteq \exists S.N$ be the axiom

chosen in (S5). Moreover, let $d = \pi_{q_i}(x)$. Due to step (S8) in the rewriting and the fact that $\mathcal{I}_\mathcal{O}$ is a model of $\mathcal{O}$, we have $d \in (\exists S.N)^{\mathcal{I}_\mathcal{O}}$. Then there is $d' \in \Delta^{\mathcal{I}_\mathcal{O}}$ such that $(d, d') \in S^{\mathcal{I}_\mathcal{O}}$ and $d' \in N^{\mathcal{I}_\mathcal{O}}$. Define the mapping $\pi_{q_{i-1}}$ for the variables of $q_i{-}1$ as follows: (a) $\pi_{q_{i-1}}(z) = d'$ for all variables $z \in V_\ell$, (b) $\pi_{q_{i-1}}(u) = d$ for all variables $u \in V_p$, and (c) $\pi_{q_{i-1}}(z) = \pi_{q_i}(z)$ for the remaining variables z. Then $\pi_{q_{i-1}}$ is a match for q_{i-1} in $\mathcal{I}_\mathcal{O}$. To see this, assume an atom α in q_{i-1}. We show that $\pi_{q_{i-1}}$ makes α true in $\mathcal{I}_\mathcal{O}$. There can be two possibilities:

(i) α has an occurrence of a variable from V_ℓ. In this case we have 3 more possibilities:

a) α is a unary atom of the form $\alpha = A(z)$. Then $z \in V_\ell$ and $\pi_{q_{i-1}}(z) = d'$ by construction of $\pi_{q_{i-1}}$. As noted above, $d' \in N^{\mathcal{I}_\mathcal{O}}$. By (S5.b) we have $A \in N$.

b) α is a binary atom $\alpha = r(y, x)$, where $y \in V_p$. We know $\pi_{q_{i-1}}(y) = d$ and $\pi_{q_{i-1}}(x) = d'$. As noted above, $(d, d') \in S^{\mathcal{I}_\mathcal{O}}$. By (S5.b) we have $r \in S$.

c) α is a binary atom $\alpha = r(y, x)$, where $y \in V_\ell$. We know $\pi_{q_{i-1}}(y) = \pi_{q_{i-1}}(x) = d'$. By (S5.c), there is a transitive $s \sqsubseteq^*_\mathcal{T} r$ such that $\{s, s^-\} \subseteq S$, or there is an axiom $M' \sqsubseteq \exists S'.N' \in \Xi(\mathcal{T}^*)$ such that $M' \subseteq N$ and $\{s, s^-\} \subseteq S'$. Since $\mathcal{I}_\mathcal{O}$ is a model of $\mathcal{O}$, we have $(d', d') \in r^{\mathcal{I}_\mathcal{O}}$.

(ii) α does not have an occurrence of a variable from V_ℓ. We distinguish the following cases:

a) α has no variables from V_p. Then $\alpha \in body(q_i)$ and the claim follows from (c) in the definition of $\pi_{q_{i-1}}$.

b) α is a unary atom $\alpha = A(u)$ with $u \in V_p$, which was replaced by $A(x)$ in (S7). By construction of $\pi_{q_{i-1}}$ we have $\pi_{q_{i-1}}(u) = d$. As $A(x) \in body(q_i)$, we have that $\pi_{q_i}(x) = d$ implies $d \in A^{\mathcal{I}_\mathcal{O}}$ as desired.

c) α is a binary atom $\alpha = r(u, z)$ with $u \in V_p$ and $z \notin V_p$, which was replaced by $r(x, z)$ in (S7). Since π_{q_i} is a match for q_i in $\mathcal{I}_\mathcal{O}$ and $r(x, z) \in body(q_i)$, π_{q_i} satisfies $r(x, z)$. By construction of $\pi_{q_{i-1}}$ we have $\pi_{q_{i-1}}(u) = \pi_{q_i}(x) = d$ and $\pi_{q_{i-1}}(z) = \pi_{q_i}(z)$, hence π_{q_i} satisfies $r(u, z)$.

d) the cases $\alpha = r(z, u)$ with either $u \in V_p$ and $z \notin V_p$, or $\{z, u\} \subseteq V_p$, are both analogous to the previous one.

We show $ans^\mathcal{T}(\mathcal{I}_\mathcal{O}, q) \subseteq ans(\mathcal{J}, \mathsf{rew}_\mathcal{T}(q))$. To show this we need some book-keeping when chasing $\mathcal{J}$ w.r.t. $\Xi(\mathcal{T}^*)$. We prescribe the naming of fresh domain elements introduced during the chase procedure. In particular, if d is a successor of e according to Definition 4.36, then d is an expression of the form $e \cdot n$, where n is a integer. For $d \in \Delta^\mathcal{J}$, let $|d| = 0$. For the elements $w \cdot n \in \Delta^{\mathcal{I}_\mathcal{O}}$, let $|w \cdot n| = |w| + 1$.

Suppose $h(\vec{x})$ is the head atom of q. Assume a tuple $\vec{u} \in ans^\mathcal{T}(\mathcal{I}_\mathcal{O}, q)$. By definition, there is match π_q for q in $\mathcal{I}_\mathcal{O}$ such that $\vec{u} = \pi_q(\vec{x})$. We have to show that there exists $q' \in \mathsf{rew}_\mathcal{T}(q)$ and a match $\pi_{q'}$ for q' in $\mathcal{J}$ such that $\vec{u} = \pi_{q'}(\vec{x})$. For any match π' in $\mathcal{I}_\mathcal{O}$, let

$$deg(\pi') = \sum_{y \in rng(\pi')} |\pi'(y)|.$$

Then, given that $q \in \mathsf{rew}_{\mathcal{T}}(q)$, to prove the claim it suffices to prove the following statement: if $q_1 \in \mathsf{rew}_{\mathcal{T}}(q)$ has a match π_{q_1} for q_1 in $\mathcal{I}_{\mathcal{O}}$ such that $\vec{u} = \pi_{q_1}(\vec{x})$ and $deg(\pi_{q_1}) > 0$, then there exists $q_2 \in \mathsf{rew}_{\mathcal{T}}(q)$ that has a match π_{q_2} for q_2 in $\mathcal{I}_{\mathcal{O}}$ such that $\vec{u} = \pi_{q_2}(\vec{x})$ and $deg(\pi_{q_2}) < deg(\pi_{q_1})$.

Assume $q_1 \in \mathsf{rew}_{\mathcal{T}}(q)$ as above. Since $deg(\pi_{q_1}) > 0$ by assumption, there must exists a variable x of q_1 such that $\pi_{q_1}(x) \notin \mathsf{N_I}$. Take such an x for which there is no variable x' of q_1 with $\pi_{q_1}(x)$ being a prefix of $\pi_{q_1}(x')$. That is, there is no variable x' of q_1 with $\pi_{q_1}(x') = \pi_{q_1}(x) \cdot w$ for some w. Intuitively, the image of π_{q_1} induces a subforest in $\mathcal{I}_{\mathcal{O}}$; the variable x is mapped into a leaf node in this forest.

Let $d_x = \pi_{q_1}(x)$, and d_p be the parent element of d_x, i.e. $d_x = d_p \cdot n$ for some integer n. We know from the construction of $\mathcal{I}_{\mathcal{O}}$ that d_x was introduced by an application of an axiom $ax = M \sqsubseteq \exists S.N \in \Xi(\mathcal{T}^*)$ such that $d_p \in M^{\mathcal{I}_{\mathcal{O}}}$. We take a query q_2 obtained from q_1 as follows:

- For Step (S1) choose $V_\ell = \{y \in var(q_1) \mid \pi_{q_1}(y) = d_x\}$ (note that since $d_x \notin \mathsf{N_I}$, all such y are non-distinguished).

- For Step (S3) let $\Gamma = \{s(y,x) \in q_1 \mid x \in V_\ell \wedge \pi_{q_1}(y) \neq d_x \wedge \pi_{q_1}(y) \neq d_p\}$ be the set of atoms we choose to rewrite. Note that due to the selection of the atoms in Γ and since π_{q_1} is a $\mathcal{T}$-match for q_1, by definition of $\mathcal{T}$-matches, for every atom $s(y,x) \in \Gamma$ there exists a transitive role r_s with $r_s \sqsubseteq^*_{\mathcal{T}} s$ such that there is an r_s-path from $\pi_{q_1}(y)$ to $\pi_{q_1}(x)$. Using this role r_s, we rewrite $s(y,x)$ into $r_s(y,u), r_s(u,x)$. Observe that, since d_p is the parent of d_x in $\mathcal{I}_{\mathcal{O}}$ and $\pi_1(y) \neq d_p$, then d_p is in the r_s-path from $\pi_{q_1}(y)$ to $\pi_{q_1}(x)$ and the following holds:

  (†) there is an r_s-path from $\pi_{q_1}(y)$ to d_p, and $(d_p, d_x) \in {r_s}^{\mathcal{I}_{\mathcal{O}}}$.

  Observe also that if $\Gamma \neq \emptyset$, then in Step (S4) we get that $u \in V_p$.

- For Step (S5), choose ax given above. To see that (S5.a) holds, take any $r(y,x)$ where $x \in V_\ell$ and $y \in V_p$. We have to show $r \in S$, and we have two cases:

  i. $y \in var(q_1)$ and $\pi_{q_1}(y) = d_p$. Since π_{q_1} is a $\mathcal{T}$-match for q_1 and d_x is a successor of d_p, we must have $(\pi_{q_1}(y), \pi_{q_1}(x)) \in r^{\mathcal{I}_{\mathcal{O}}}$. Then due to the construction of $\mathcal{I}_{\mathcal{O}}$, $r \in S$.

  ii. if $y = u$ is the fresh variable introduced in Step (S3), then r is the role r_s chosen above and by (†) we have $(d_p, d_x) \in r^{\mathcal{I}_{\mathcal{O}}}$, which implies $r \in S$ due to the construction of $\mathcal{I}_{\mathcal{O}}$.

  To see that (S5.b) holds, take any $A(z)$ where $z \in V_\ell$. We have to show $A \in N$. Since π_{q_1} is a $\mathcal{T}$-match for q_1, we have $\pi_{q_1}(z) \in A^{\mathcal{I}_{\mathcal{O}}}$. Since $\pi_{q_1}(z) = d_x$, by construction of $\mathcal{I}_{\mathcal{O}}$ we have $A \in N$.

  Finally, we check that (S5.c) holds. Take an atom $r(x,y)$ in q_1 such that $x, y \in V_\ell$. Since $\pi_{q_1}(z) = \pi_{q_1}(x) = d_x$ and π_{q_1} is a $\mathcal{T}$-match, we have a "self-loop" from d_x to itself, that is, there is a transitive $s \sqsubseteq^*_{\mathcal{T}} r$ and an s-path from d_x to d_x. This path must pass through a domain element $d \neq d_x$, an in particular it muss pass a d that is either the parent d_x or some child of d_x. Due to the construction of $\mathcal{I}_{\mathcal{O}}$, (i.) is satisfied in the former case and (ii.) is satisfied in the latter case.

Finally, a match π_{q_2} for q_2 in $\mathcal{I}_\mathcal{O}$ such that $\vec{u} = \pi_{q_2}(\vec{x})$ and $deg(\pi_{q_2}) < deg(\pi_{q_1})$ is obtained from π_{q_1} by setting (a) $\pi_{q_2}(z) = \pi_{q_1}(z)$ for all z of q_2 with $z \neq x$, and (b) $\pi_{q_2}(x) = d_p$. It is easy to check that π_{q_2} is a $\mathcal{T}$-match for q_2 because q_2 is intuitively a subquery of q_1. Observe that $vars(q_2) \subseteq vars(q_1)$ because any new variable introduced in Step (S3) is eliminated in Step (S7). Hence, $deg(\pi_{q_2}) < deg(\pi_{q_1})$ follows from the fact that (i) $|\pi_{q_2}(z)| = |\pi_{q_1}(z)|$ for all z of q_2 with $z \neq x$, and (ii) $|\pi_{q_2}(x)| = |\pi_{q_1}(x)| - 1$.

□

By the above reduction, we can answer q over $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ by posing $\mathsf{rew}_\mathcal{T}(q)$ over the Datalog program $\mathcal{A} \cup \mathsf{cr}(\mathcal{T}^*)$.

5.3.3 Rewriting Positive Weakly DL-safe KBs

The rewriting method for CQs over Horn-$\mathcal{SHIQ}$ ontologies can be naturally generalized to the more general positive weakly DL-safe KBs.

Positive Weakly DL-safe KBs

Following [LR98] we now define positive weakly DL-safe knowledge bases (KBs for short in this section). Let $\mathsf{N_I}$, $\mathsf{N_V}$ and $\mathsf{N_D}$ be countable infinite sets of *constants* (or, *individuals*), *variables* and Datalog *relations*, respectively; we assume these sets as well as $\mathsf{N_C}$ and $\mathsf{N_R}$ are all mutually disjoint. A (positive weakly DL-safe) KB is a triple $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{P})$, where $\mathcal{T}$ is a TBox and $\mathcal{P}$ is a program of rules in the following format which are weakly DL-safe (see Definition 3.18).

$$h(\vec{u}) \leftarrow p_1(\vec{v_1}), \ldots, p_m(\vec{v_m}), \tag{5.18}$$

where $h(\vec{u})$ is an atom (the *head*), $p_i \in \mathsf{N_C} \cup \mathsf{N_R} \cup \mathsf{N_D}$, $\{p_1(\vec{v_1}), \ldots, p_m(\vec{v_m})\}$ are also atoms (the *body* atoms, denoted $body(\rho)$), and $\vec{u}, \vec{v_1}, \ldots, \vec{v_m}$ are tuples of variables.

The semantics for a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{P})$ is given by extending an interpretation $\mathcal{I}$ to symbols in $\mathsf{N_I} \cup \mathsf{N_D}$. For any $c \in \mathsf{N_I}$ and $p \in \mathsf{N_D}$ of arity n, we have $c^\mathcal{I} \in \Delta^\mathcal{I}$ and $p^\mathcal{I} \subseteq (\Delta^\mathcal{I})^n$. A *match* π for a rule ρ of the form (5.18) in $\mathcal{I}$ is a mapping from variables in ρ to elements in $\Delta^\mathcal{I}$ such that $\pi(\vec{t}) \in p^\mathcal{I}$ for each body atom $p(\vec{t})$ of ρ. We define:

(a) $\mathcal{I} \models \rho$ if $\pi(\vec{u}) \in h^\mathcal{I}$ for every match π for ρ in $\mathcal{I}$,

(b) $\mathcal{I} \models \mathcal{P}$ if $\mathcal{I} \models \rho$ for each $\rho \in \mathcal{P}$,

(c) $\mathcal{I} \models \mathcal{A}$ if $(\vec{c})^\mathcal{I} \in p^\mathcal{I}$ for all $p(\vec{c}) \in \mathcal{A}$,

(d) $\mathcal{I} \models \mathcal{K}$ if $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \mathcal{A}$ and $\mathcal{I} \models \mathcal{P}$.

Finally, given a ground atom $p(\vec{c})$, $\mathcal{K} \models p(\vec{c})$ if $(\vec{c})^\mathcal{I} \in p^\mathcal{I}$ for all models $\mathcal{I}$ of $\mathcal{K}$. We remark that weakly DL-safe KBs are a natural extension of conjunctive queries.

Reducing positive weakly DL-safe KBs to Datalog

The ground atomic consequences of $\mathcal{K}$ can be collected by fixed-point computation: until no new consequences are derived, pose rules in $\mathcal{P}$ as CQs over $(\mathcal{T}, \mathcal{A})$ and put the obtained answers into $\mathcal{A}$. If we employ the rewriting in Definition 5.9, this computation can achieved using a plain Datalog program.

Theorem 5.13. *For a ground atom α over a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{P})$ where $\mathcal{T}$ is a Horn-$\mathcal{SHIQ}$ TBox and $\mathcal{P}$ is positive weakly DL-safe, we have $(\mathcal{T}, \mathcal{A}, \mathcal{P}) \models \alpha$ iff $\mathsf{cr}(\mathcal{T}^*) \cup \mathsf{rew}_{\mathcal{T}}(\mathcal{P}) \cup \mathcal{A} \models \alpha$.*

Proof. First of all, let $\mathcal{K}_1 \models^{\mathcal{T}} \alpha_1$ be defined as $\mathcal{K}_1 \models \alpha_1$ but using the notion of a $\mathcal{T}$-match instead of a (plain) match. Since $(\mathcal{T}, \mathcal{A}, \mathcal{P}) \models \alpha$ iff $(\mathcal{T}^*, \mathcal{A}, \mathcal{P}) \models^{\mathcal{T}} \alpha$, it suffices to show $(\mathcal{T}^*, \mathcal{A}, \mathcal{P}) \models^{\mathcal{T}} \alpha$ iff $\mathsf{cr}(\mathcal{T}^*) \cup \mathsf{rew}_{\mathcal{T}}(\mathcal{P}) \cup \mathcal{A} \models \alpha$.

Let $\mathcal{P}' = \mathsf{cr}(\mathcal{T}^*) \cup \mathsf{rew}_{\mathcal{T}}(\mathcal{P})$.

For the "if" direction, the interesting case is where $(\mathcal{T}^*, \mathcal{A})$ is consistent. Note first that $(\mathcal{T}^*, \mathcal{A}, \mathcal{P}) \models^{\mathcal{T}} \alpha'$ for all $\alpha' \in \mathcal{A}$. Hence, intuitively, it suffices to show that the rules of $\mathcal{P}'$ applied on $\mathcal{A}$ derive consequences of $(\mathcal{T}^*, \mathcal{A}, \mathcal{P})$. In particular, assume a rule

$$r = h(\vec{u}) \leftarrow b_1(\vec{v_1}), \ldots, b_m(\vec{v_m})$$

in $\mathcal{P}'$ and take a mapping $\pi : vars(r) \rightarrow \mathsf{N_I}$. To prove the claim it suffices to show that $(\mathcal{T}^*, \mathcal{A}, \mathcal{P}) \models^{\mathcal{T}} b_1(\pi(\vec{v_1})), \ldots, (\mathcal{T}^*, \mathcal{A}, \mathcal{P}) \models^{\mathcal{T}} b_m(\pi(\vec{v_m}))$ implies $(\mathcal{T}^*, \mathcal{A}, \mathcal{P}) \models^{\mathcal{T}} h(\pi(\vec{u}))$.

The statement is straightforward if r is a rule in $\mathsf{cr}(\mathcal{T}^*)$, because $\mathsf{cr}(\mathcal{T}^*)$ encodes a subset of $\Xi(\mathcal{T}^*)$, which contains only logical consequences of $\mathcal{T}^*$.

Suppose $r \in \mathsf{rew}_{\mathcal{T}}(r')$, for some rule $r' \in \mathcal{P}$. Let $\mathcal{K}' = (\mathcal{T}^*, \mathcal{A}', \mathcal{P})$, where

$$\mathcal{A}' = \mathcal{A} \cup \{b_1(\pi(\vec{v_1})), \ldots, b_m(\pi(\vec{v_m})\}.$$

By applying Theorem 5.12, we get $h(\pi(\vec{u})) \in ans((\mathcal{T}^*, \mathcal{A}'), r')$. Hence, $\mathcal{K}' \models^{\mathcal{T}} h(\pi(\vec{u}))$. Since $\mathcal{K}' \equiv (\mathcal{T}^*, \mathcal{A}, \mathcal{P})$ due to the induction hypothesis, we also get $(\mathcal{T}^*, \mathcal{A}, \mathcal{P}) \models^{\mathcal{T}} h(\pi(\vec{u}))$.

We prove the "only if" direction. The only interesting case is where $\mathsf{cr}(\mathcal{T}^*) \cup \mathsf{rew}_{\mathcal{T}}(\mathcal{P}) \cup \mathcal{A}$ is consistent. In this case, it suffices to show the existence of a model $\mathcal{I}$ of $(\mathcal{T}^*, \mathcal{A}, \mathcal{P})$ such that $\mathcal{I} \not\models \alpha$ for all α such that $\mathsf{cr}(\mathcal{T}^*) \cup \mathsf{rew}_{\mathcal{T}}(\mathcal{P}) \cup \mathcal{A} \not\models \alpha$. Let $\mathcal{A}'$ be the set of all ground α such that $\mathsf{cr}(\mathcal{T}^*) \cup \mathsf{rew}_{\mathcal{T}}(\mathcal{P}) \cup \mathcal{A} \models \alpha$. We let $\mathcal{I} = chase(\mathcal{A}', \Xi(\mathcal{T}^*))$. Since the chase procedure does change the ground atoms that are entailed, $\mathcal{I} \not\models \alpha$ for all α such that $\mathsf{cr}(\mathcal{T}^*) \cup \mathsf{rew}_{\mathcal{T}}(\mathcal{P}) \cup \mathcal{A} \not\models \alpha$. It only remains to see that

(a) $\mathcal{I} \models (\mathcal{T}^*, \mathcal{A})$. Due to consistency of $\mathsf{cr}(\mathcal{T}^*) \cup \mathsf{rew}_{\mathcal{T}}(\mathcal{P}) \cup \mathcal{A}$, we also have that $\mathsf{cr}(\mathcal{T}^*) \cup \mathcal{A}'$ is consistent. Due to Theorem 4.37, it suffices to show that $\mathcal{A}' = MM(\mathsf{cr}(\mathcal{T}^*) \cup \mathcal{A}')$. Trivially, $\mathcal{A}' \subseteq MM(\mathsf{cr}(\mathcal{T}^*) \cup \mathcal{A}')$. For $\mathcal{A}' \supseteq MM(\mathsf{cr}(\mathcal{T}^*) \cup \mathcal{A}')$, assume there is $\beta \in MM(\mathsf{cr}(\mathcal{T}^*) \cup \mathcal{A}')$ with $\beta \notin \mathcal{A}'$. Then β is derived via a rule $r \in \mathsf{cr}(\mathcal{T}^*)$ using some match π in $\mathcal{A}'$. Then it must be the case that $\beta \in \mathcal{A}'$ because by construction $\mathcal{A}'$ is closed under the rules in $\mathsf{cr}(\mathcal{T}^*)$.

(b) $\mathcal{I} \models \mathcal{P}$. Assume a rule $r \in \mathcal{P}$ with a mapping π from variables of r to $\Delta^{\mathcal{I}}$ such that $\mathcal{I} \models b(\pi(\vec{v}))$ for each body atom $b(\vec{v})$ of r. We have to show that $\mathcal{I} \models h(\pi(\vec{u}))$, where $h(\vec{u})$ is the

head of r. Due to weak DL-safety of $\mathcal{P}$, $\pi(x) \in \mathsf{N_I}$ for each variable x in $\vec{u}$. In other words, π is an ordinary match for a conjunctive query. In particular, $\pi(\vec{v}) \in ans((\mathcal{T}^*, \mathcal{A}'), r)$ since $\mathcal{A}' = MM(\mathsf{cr}(\mathcal{T}^*) \cup \mathcal{A}')$. Then due to Theorem 5.12, we have a match π' for some $r' \in \mathsf{rew}_\mathcal{T}(\mathcal{P})$ in $\mathcal{A}'$. Since $\mathcal{A}'$ is closed under rules in $\mathsf{rew}_\mathcal{T}(\mathcal{P})$, we have $h(\pi(\vec{u})) \in \mathcal{A}'$ and thus $\mathcal{I} \models h(\pi(\vec{u}))$.

□

The algorithm obtained by the above reduction is worst-case optimal in terms of combined and data complexity.

Theorem 5.14. *For a ground atom α over a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{P})$ where $\mathcal{T}$ is a Horn-$\mathcal{SHIQ}$ TBox and $\mathcal{P}$ is weakly DL-safe, checking $(\mathcal{T}, \mathcal{A}, \mathcal{P}) \models \alpha$ is* EXPTIME*-complete in general, and* P*-complete when only the size of $\mathcal{A}$ is counted (i.e. in data complexity).*

Proof. By Theorem 5.13, checking $(\mathcal{T}, \mathcal{A}, \mathcal{P}) \models \alpha$ is equivalent to deciding $\mathsf{cr}(\mathcal{T}^*) \cup \mathsf{rew}_\mathcal{T}(\mathcal{P}) \cup \mathcal{A} \models \alpha$. We analyze the computational cost of the latter check.

We first recall that $\Xi(\mathcal{T}^*)$ can be computed in exponential time in size of $\mathcal{T}$ and is independent from $\mathcal{A}$: the calculus in Table 4.4 only infers axioms of the form $M \sqsubseteq B$ and $M \sqsubseteq \exists S.N$, where M, N are conjunctions of atomic concepts, B is atomic and S is a conjunction of roles. The number of such axiom is single exponential in the size of $\mathcal{T}$.

Observe that $\mathsf{rew}_\mathcal{T}(\mathcal{P})$ is finite and computable in time exponential in the size of $\mathcal{T}$ and $\mathcal{P}$: rules in $\mathsf{rew}_\mathcal{T}(\rho)$, where $\rho \in \mathcal{P}$, use only relation names and variables that occur in ρ and $\mathcal{T}$ (fresh variables introduced in (S3) are eliminated in (S6) and (S7)). Hence, the size of each rule resulting from a rewrite step is of size polynomial in the size of $\mathcal{T}$ and $\mathcal{P}$, and thus the number of rules in $\mathsf{rew}_\mathcal{T}(\mathcal{P})$ is at most exponential in the size of $\mathcal{T}$ and $\mathcal{P}$. The size of $\mathsf{rew}_\mathcal{T}(\mathcal{P})$ is constant when data complexity is considered.

Furthermore, the *grounding* of $\mathsf{cr}(\mathcal{T}^*) \cup \mathsf{rew}_\mathcal{T}(\mathcal{P}) \cup \mathcal{A}$ is exponential in the size of $\mathcal{K}$, but polynomial for fixed $\mathcal{T}$ and $\mathcal{P}$. By the complexity of Datalog, it follows that the algorithm resulting from Theorem 5.13 is exponential in combined but polynomial in data complexity.

The above complexity result is worst-case optimal, and applies already to plain conjunctive queries [Eit+08c]. □

5.4 CQ-Programs

Reduction of CQ-Programs to Datalog$^\neg$

The reduction of CQ-Programs to Datalog$^\neg$ is similar to the reduction of DL-Programs to Datalog$^\neg$.

We first extend the the notion of Datalog rewritability for instance query (Definition 4.1) to conjunction query.

Definition 5.15. *A Description Logic $\mathcal{DL}$ is* Datalog-rewritable *for conjunctive query if there exists a transformation $\Phi_{\mathcal{DL}}$ from $\mathcal{DL}$ KBs to* Datalog *programs such that, for any $\mathcal{DL}$ KB Σ and CQ $\alpha = q(\mathbf{X}) \leftarrow \beta(\mathbf{X}, \mathbf{Y})$ over Σ,*

(i) $\Sigma \models \alpha(\mathbf{o})$ *iff* $\Phi_{\mathcal{DL}}(\Sigma, \alpha) \models q(\mathbf{o})$ *for any individual(s)* $\mathbf{o}$ *from* Σ*;*

(ii) $\Phi_{\mathcal{DL}}$ *is* modular*, i.e., for* $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ *where* $\mathcal{T}$ *is a TBox and* $\mathcal{A}$ *an ABox,* $\Phi_{\mathcal{DL}}(\Sigma, \alpha) = \Phi_{\mathcal{DL}}(\mathcal{T}, \alpha) \cup \mathcal{A}$*;*

Clearly, as instance query of concept or role name Q can be seen as a conjunctive query $q(\mathbf{X}) \leftarrow Q(\mathbf{X})$, the above definition is a proper extension of Definition 4.1. It is easy to verify that DLs $\mathcal{LDL}^+$ and Horn-$\mathcal{SHIQ}$ are Datalog-rewritable for conjunctive query.

Definition 5.16. *The translation of CQ-Programs* (Σ, P) *to a* Datalog$^\neg$ *program is then built up of the following four components:*

- *A* Datalog *program* $\bigcup_{DL[\lambda;\alpha] \text{ in } P} [\Phi_{\mathcal{DL}}(\Sigma, \alpha)]_{\lambda,\alpha}$ *where* $[\cdot]_{\lambda,\alpha}$ *is obtained by subscripting allq ip concept and role names inside with* λ, α*.*

- *A* Datalog *program* $\rho(P)$ *containing the rules* $S_{i,\lambda,\alpha}(\mathbf{X_i}) \leftarrow p_i(\mathbf{X_i})$*,* $1 \leq i \leq m$*, for all* $DL[\lambda; \alpha]$ *in* P *and* $\lambda = S_1 \uplus p_1, \ldots, S_m \uplus p_m$ *where the arity of* $\mathbf{X_i}$ *matches the one of* S_i*.*

- *A set* T_P *of* Datalog *facts* $\top(a)$ *and* $\top^2(a, b)$ *for all* a, b *in the Herbrand domain of* P *to ensure their introduction in* Σ*.*

- *Finally,* P^{ord} *results from replacing each dl-atom* $DL[\lambda; \alpha](\mathbf{t})$ *in* P *with a new atom* $q_{\lambda,\alpha}(\mathbf{t})$*.*

The transformation of the dl*-program* $\mathcal{KB}$ *is then defined as*

$$\Omega(\mathcal{KB}) = \bigcup_{DL[\lambda;\alpha] \text{ in } P} [\Phi_{\mathcal{DL}}(\Sigma, \alpha)]_{\lambda,\alpha} \cup P^{ord} \cup \rho(P) \cup T_P \tag{5.19}$$

Analogous to Theorem 4.7 and Theorem 4.9, we establish the correctness of $\Omega(KB)$:

Theorem 5.17. *Let* $\mathcal{KB} = (L, P)$ *be a* cq*-program over a* Datalog*-rewritable for CQ DL. Then the answer sets of* $\mathcal{KB}$ *correspond 1-1 to the answer sets of* $\Omega(\mathcal{KB})$*:*

(i) every answer set of $\mathcal{KB}$ *is extendible to an answer set of* $\Omega(\mathcal{KB})$*; and*

(ii) for every answer set J *of* $\Omega(\mathcal{KB})$*, its restriction* $I = J|_{HB_P}$ *to* HB_P *is an answer set of* $\mathcal{KB}$*.*

Theorem 5.18. *Let* $\mathcal{KB} = (L, P)$ *be a* cq*-program over a* Datalog*-rewritable for CQ DL and* a *be a ground atom from* HB_P*. Then,*
$\mathcal{KB} \models^{wf} a$ *iff* $\Omega(\mathcal{KB}) \models^{wf} a$*.*

We can prove the above two theorems using exactly the same techniques in the proofs of theorems 4.7 and 4.9 by adapting the notion I^{Ψ} to I^{Ω} as following.

For a cq-program $\mathcal{KB} = (\Sigma, P)$ over a Datalog-rewritable for CQ DL and an interpretation I over HB_P, we define an interpretation I^{Ω} for $\Omega(\mathcal{KB})$:

$$I^{\Omega} = I \cup \bigcup_{DL[\lambda;\alpha]\in P} \mathsf{MM}(\Omega(\Sigma_{\lambda,\alpha} \cup S(\lambda, \alpha, I)))$$

where

$$S(\lambda, \alpha, I) = \{S_{\lambda,\alpha}(\mathbf{c}) \mid S \uplus p \in \lambda, p(\mathbf{c}) \in I\}.$$

We show a concrete example of reducing CQ-Programs over Horn-$\mathcal{SHIQ}$ ontologies to Datalog$^{\neg}$ program.

Example 5.19. *Let CQ-Program $\mathcal{KB} = (L, P)$, where $L = (\mathcal{T}, \mathcal{A})$, $\mathcal{T} = \{A \sqsubseteq \exists R.B\}$, $\mathcal{A} = \{R(a, b), A(c), A(b), D(a)\}$, and*

$$P = \left\{ \begin{array}{l} p(x) \leftarrow DL[D \uplus s; D(x), R(x, y), B(y)](x), not\ DL[D](x) \\ s(a). \quad s(b). \quad s(c). \end{array} \right\}.$$

As the ontology L is in Horn-$\mathcal{SHIQ}$, we could reduce the CQ-Program $\mathcal{KB}$ to a Datalog$^{\neg}$ *program using the query rewriting for Horn-$\mathcal{SHIQ}$.*

Let $\lambda_1 = D \uplus p, \lambda_2 = \epsilon$, $\alpha_1 = q(x) \leftarrow D(x), R(x, y), B(y)$, $\alpha_2 = q(x) \leftarrow D(x)$,

- *We apply the query rewriting algorithm of Horn-$\mathcal{SHIQ}$ for CQs in the DL-atoms.*

  The rules for α_1 in $DL[\lambda_1; \alpha_1]$ are

$$\Phi(L, \alpha_1) = \left\{ \begin{array}{l} q(x) \leftarrow D(x), R(x, y), B(y) \\ q(x) \leftarrow D(x), A(x) \\ R(a, b). \quad A(c). \quad A(b). \quad D(a). \end{array} \right\}.$$

  The subscripted version is

$$[\Phi(L, \alpha_1)]_{\lambda_1,\alpha_1} = \left\{ \begin{array}{l} q_{\lambda_1,\alpha_1}(x) \leftarrow D_{\lambda_1,\alpha_1}(x), R_{\lambda_1,\alpha_1}(x, y), B_{\lambda_1,\alpha_1}(y) \\ q_{\lambda_1,\alpha_1}(x) \leftarrow D_{\lambda_1,\alpha_1}(x), A_{\lambda_1,\alpha_1}(x) \\ R_{\lambda_1,\alpha_1}(a, b). \quad A_{\lambda_1,\alpha_1}(c). \quad A_{\lambda_1,\alpha_1}(b). \quad D_{\lambda_1,\alpha_1}(a). \end{array} \right\}$$

  Similarly, the rules for α_1 in $DL[\lambda_2; \alpha_2]$ are

$$\Phi(L, \alpha_2) = \left\{ \begin{array}{l} q(x) \leftarrow D(x) \\ R(a, b). \quad A(c). \quad A(b). \quad D(a). \end{array} \right\},$$

  and the subscripted version is

$$[\Phi(L, \alpha_2)]_{\lambda_2,\alpha_2} = \left\{ \begin{array}{l} q_{\lambda_2,\alpha_2}(x) \leftarrow D_{\lambda_2,\alpha_2}(x) \\ R_{\lambda_2,\alpha_2}(a, b). \quad A_{\lambda_2,\alpha_2}(c). \quad A_{\lambda_2,\alpha_2}(b). \quad D_{\lambda_2,\alpha_2}(a). \end{array} \right\}$$

- *Rules $\rho(P)$:*

  For $DL[\lambda_1; \alpha_1]$, we add

  $$D_{\lambda_1,\alpha_1}(x) \leftarrow s(x).$$

  For $DL[\lambda_2; \alpha_2]$, as $\lambda_2 = \epsilon$, we do not need to add new rules.

- *Facts for introducing the individuals: $T_P = \{\top(u), \top(u, v) \mid u, v \in \{a, b, c\}\}$*

- *Finally, we have*

$$P^{ord} = \left\{ \begin{array}{l} p(x) \leftarrow DL[D \uplus s; D(x), R(x, y), B(y)](x), not\ DL[D](x) \\ s(a). \quad s(b). \quad s(c). \end{array} \right\}.$$

The result of the reduction is the $\mathsf{Datalog}^{\neg}$ *program*

$$\Omega(\mathcal{KB}) = [\Phi_{\mathcal{DL}}(L, \alpha_1)]_{\lambda_1,\alpha_1} \cup [\Phi_{\mathcal{DL}}(L, \alpha_2)]_{\lambda_2,\alpha_2} \cup P^{ord} \cup \rho(P) \cup T_P.$$

It is not difficult to check that $\mathcal{KB}$ has a single answer set $I = \{s(a), s(b), s(c), p(b)\}$ and $\Omega(\mathcal{KB})$ has a single answer set $J \supset I$; all elements in $J \setminus I$ are subscripted. The correspondence is as we expected.

5.5 Related Work

The framework of terminological default theory is proposed in [BH95], where Baader and Hollunder showed that to avoid the semantics problems and undecidability issues of embedding defaults to description logics, we have to restrict the application of Reiter's style rules to named individuals only. The first result of embedding terminological default theory to dl-programs is in [Eit+08a]. Later, more transformations and optimization techniques are developed in [DEK09] and are implemented as a front end inside the dl-plugin for the DLVHEX system. Since the generated dl-programs have recursions through the dl-atoms, they are still very challenging for DLVHEX even with optimizations. In contrast, our techniques transformation using $\mathsf{Datalog}^{\neg}$ as the targeting linage, which is much easier to evaluate on ASP engines.

We note that DL-safe CQs are a proper fragment of the more general DL-safe KBs, because DL-safe KBs have more features, e.g., multiple rules, DL-predicates in the rule head, and non-DL-predicates in the rules [HMS04]. It is worth mentioning that the general DL-safe KBs can be reduced to dl-programs [Eit+08a] by a "guess-and-check" encoding. Although the reduction in section 5.2 only works for DL-safe CQs, it is already useful in the many practical cases and is very easy to implement on top of the inline evaluation framework. Actually, DL-safe CQs have been considered and implemented in many DL reasoners (e.g., KAON2 and Pellet). Our technique is strongly related to KAON2 approach, where they use $\mathsf{Datalog}^{\vee}$ as the targeting language. Systems KAON2, HermiT, and Pellet even support SWRL rules under dl-safeness, which can be seen as another fragment of the general DL-safe KBs.

In the field of query answering over DL ontologies, since Calvanese *el al.* introduced query rewriting in their seminal work on DL-Lite [Cal+07], many query rewriting techniques have

been developed and implemented, (e.g., [PMH09], [RA10], [CTS11], [GOP11]), usually aiming at an optimized rewriting size. Some of them also go beyond $DL\text{-}Lite$; e.g. Perez-Urbina *el al.* cover $\mathcal{ELHI}$ [PMH09], while Gottlob *el al.* consider $\mathsf{Datalog}^{\pm}$[GOP11]. Most approaches rewrite a query into a (union of) CQs; Rosati and Almatelli generate a non-recursive Datalog program [RA10], while Perez-Urbina *el al.* produce a CQ for $DL\text{-}Lite$ and a (recursive) Datalog program for DLs of the $\mathcal{EL}$ family [PMH09]. Our approach can be seen as a hybrid, which rewrites a CQ into a union of CQs, but generates (possibly recursive) Datalog rules to capture the TBox.

Another comparable technique of query answering is the *combined approach* of Lutz *el al.* [LTW09]. In order to do query answering in $\mathcal{EL}$ with off-the-shelf RDBMSs, the authors expand the data in the ABox 'materializing' a part of the canonical model that can be used for query answering after some query rewritings. Viewing our approach as a variation of the combined approach suggests an alternative query evaluation technique: we can first close the ABox under the rules in $\mathsf{cr}(\mathcal{T})$, and then evaluate the rewritten query $\mathsf{rew}(q)$ over the closed ABox.

Our query rewriting of Horn-$\mathcal{SHIQ}$ technique resembles Rosati's [Ros07] for CQs in $\mathcal{EL}$, which replaces query atoms by existential concepts, then applies some TBox saturation and translates the rewritten queries and the TBox into Datalog. The main difference is that in Rosati's technique the rewriting takes place *before* TBox saturation, resulting in an algorithm that is best-case exponential in the size of the query. This is avoided in our approach since a rewriting step occurs only if the saturated TBox has an applicable existential axiom.

Rewriting approaches for more expressive DLs are less common. The most notable exception is Hustadt *el al.*'s translation of $\mathcal{SHIQ}$ terminologies into disjunctive Datalog [HMS07], which is implemented in the KAON2 reasoner. The latter can be used to answer queries over arbitrary ABoxes, but supports only instance queries. To our knowledge, the extension of the rewriting in [PMH09] to nominals remains to be implemented [PMH10]. In [ORS10] a Datalog rewriting is used to establish complexity bounds of standard reasoning in the Horn fragments of $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$, but it does not cover CQs.

To the best of our knowledge, there is no practical algorithm and implementation for reasoning over weakly dl-safe KBs. Rosati proposed a "guess and check" algorithm for $\mathcal{DL}$+*log* [Ros06], which is only of theoretical interest. He also showed that for $DL\text{-}Lite$+*log*, the data complexity does not increase with respect to the data complexity of the Datalog program alone. However, the optimization and implementation was left for future work. Our algorithm for weakly dl-safe KBs naturally extends the rewriting techniques for CQ and is not difficult to implement.

The framework of cq-programs are proposed in [Eit+08b], and are implemented in the DLVHEX system. We note that because the DL engine RacerPro used in DLVHEX does not fully support conjunctive queries, the results of dlvhex for cq-programs can be incomplete. Regarding the performance, as usual, we expected our inline evaluation approach over cq-programs is more efficient than the approach in DLVHEX.

CHAPTER 6

The DReW System

Our primary motivation of this thesis is to develop an efficient and practical system for reasoning over hybrid knowledge base, with a focus on dl-programs. To validate the efficiency of the inline evaluation strategy, we implemented most of the techniques developed in the chapter 4 and 5 in a new system DReW. In this chapter, we describe the implementation details of the DReW system and some applications.

6.1 System Overview

The DReW (Datalog ReWriter) system[1] is a reasoning engine for dl-programs. The system implements the inline evaluation framework of dl-programs (Section 4.1), over OWL 2 RL (Section 4.2) and OWL 2 EL (Section 4.3), and additionally support terminological default theory over OWL 2 EL (Section 5.1) and conjunctive queries under DL-safeness condition over OWL 2 RL and OWL 2 EL (Section 5.2). The techniques for conjunctive queries over Horn-$\mathcal{SHIQ}$ are implemented in another system clipper [Eit+12b; Eit+12c]. Finally, the technique for CQ-programs over Horn-$\mathcal{SHIQ}$ (Section 5.4) is not implemented yet because of the time constrains and of low priority for the moment.

The DReW system is open sourced and publicly hosted in a git repository at GitHub [2].

6.1.1 Architecture

Fig. 6.1 shows a schematic overview of the components of DReW in charge of reasoning with dl-programs by Datalog$^{\neg}$ rewriting.

- The *Ontology parser* and *DL-Rules parser* together parse the input dl-programs into the internal representation.

[1] http://www.kr.tuwien.ac.at/research/systems/drew/
[2] https://github.com/ghxiao/drew

- The main component *DL-Program Rewriter* is responsible for rewriting the dl-programs into Datalog$^\neg$ programs based on a *DL Profile* (e.g. EL or RL).

  - The *DL Rewriter* rewrites the ontology into a set of Datalog rules according to the input *DL Profile*.
  - The *DL-Atom Extractor* extracts the dl-atoms from the dl-rules.
  - The *Duplicator* generates subscripted versions of the Datalog rules form the *DL Rewriter* according to the dl-atoms.
  - The *DL-Rules Rewriter* rewrite the dl-rules to a normal Datalog$^\neg$ program by replacing the dl-atoms in the dl-programs to an ordinary Datalog predicates.
  - The *DL-Atom Rewriter* generates datalog rules simulating the dl-atoms.
  - The *Datalog Generator* combines the input dl-rules into a single Datalog$^\neg$ program.

- Finally, the *Model Builder* calls an external Datalog$^\neg$ reasoner to compute the answer sets (or well-founded models) of the generated Datalog$^\neg$ program, and converts them to the models of the original dl-program.

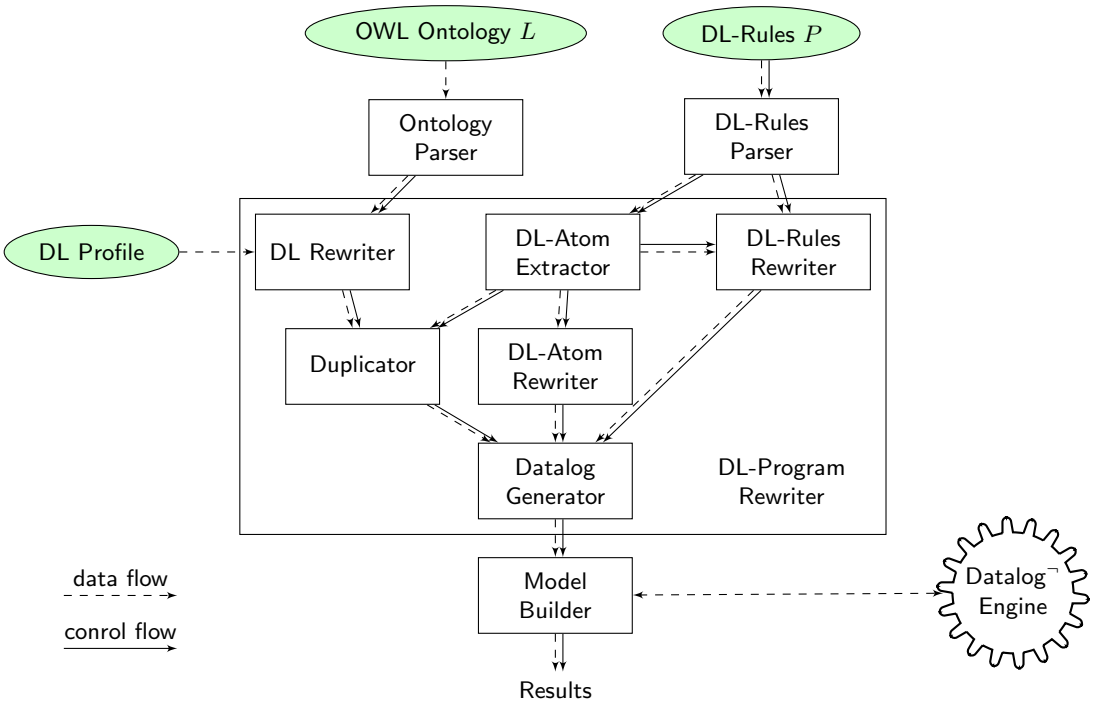


Figure 6.1: Architecture of the DReW System

Supported DLs

The current version of DReW supports ontologies in the OWL 2 RL and OWL 2 EL. The rewritings are based on the Datalog rewriting algorithms described in Section 4.2 and 4.3. To comply with the OWL 2 standard, we adapted these rewriting techniques.

OWL 2 RL rewriter is based on the algorithm for $\mathcal{LDL}^+$ (Section 4.2). Since some of the constructors in $\mathcal{LDL}^+$ (role conjunctions, transitive closure and role nominals) are not in OWL 2, we did not support these features in DReW. Actually, in the first implementation of DReW [XHE10], we supported full $\mathcal{LDL}^+$ fragment by using a modified version of OWL API. However, these non-OWL constructions are dropped in the later versions of DReW for compatibility.

OWL 2 EL rewriter is based on the algorithm for $\mathcal{SROEL}(\sqcap, \times)$ (Section 4.3). As role conjunction ($\sqcap$) and concept production ($\times$) are not available in OWL 2 EL, we have to adapt them from P_{inst}. Specifically, the following rules with $subRConj(\sqcap)$ and $subProd(\times)$ are dropped from P_{inst} in the implementation:

$$
\begin{aligned}
triple(X, W, X') &\leftarrow subRConj(V_1, V_2, W), triple(X, V_1, X'), triple(X, V_2, X') \\
self(X, W) &\leftarrow subRConj(V_1, V_2, W), self(X, V_1), self(X, V_2) \\
triple(X, W, X') &\leftarrow subProd(Y_1, Y_2, W), isa(X', Y_2), isa(X, Y_1) \\
self(X, W) &\leftarrow subProd(Y_1, Y_2, W), isa(X, Y_2), isa(X, Y_1)
\end{aligned}
$$

Concept production are seen as domain and range assertions. In the DReW system, we convert them into facts $range(V, R)$ and $domain(V, D)$ meaning that the range of V is R and the domain of V is D. The rules about the $supProd$ predicate in P_{inst}

$$
\begin{aligned}
isa(X, Z_1) &\leftarrow supProd(V, Z_1, Z_2), triple(X, V, X') \\
isa(X, Z_1) &\leftarrow supProd(V, Z_1, Z_2), self(X, V) \\
isa(X', Z_2) &\leftarrow supProd(V, Z_1, Z_2), triple(X, V, X') \\
isa(X, Z_2) &\leftarrow supProd(V, Z_1, Z_2), self(X, V)
\end{aligned}
$$

are replaced by the followings:

$$
\begin{aligned}
isa(Y, R) &\leftarrow range(V, R), triple(X, V, Y) \\
isa(X, R) &\leftarrow domain(V, R), triple(X, V, Y) \\
isa(X, R) &\leftarrow range(V, R), self(X, V) \\
isa(X, R) &\leftarrow domain(V, R), self(X, V)
\end{aligned}
$$

Using the same techniques in [Krö11], we can show that the modified calculus is complete for instance queries over $\mathcal{SROEL}$ ontologies.

Semantics

The DReW system support both the (strong) answer set semantics and well-founded semantics. The default option is the answer set semantics. The transformation preserve the models under both semantics, so as long as the underlying Datalog$^{\neg}$ engine supports both semantics, the implementation of these two semantics make no much difference.

Front Ends

Besides dl-programs, DReW also supports conjunctive query (CQ) and default reasoning front end.

Conjunctive Query front end DReW can answer conjunctive queries over DL ontologies under DL-safeness restriction based on the algorithms in Section 5.2 over OWL 2 EL and OWL 2 RL. The input query must be in the SPARQL format which corresponds to a conjunctive query, i.e. SPARQL query with only basic graph patterns.

Example 6.1. *The following CQ used in Example 5.5*

$$ans(X,Y) \leftarrow HighTrafficNode(X), wired(X,Y), HighTrafficNode(Y)$$

can be represented by a SPARQL query assuming the prefix of the URI of the predicates is "`http://www.example.org/`"*:*

```
PREFIX : <http://www.exmaple.org/>
SELECT ?X ?Y
WHERE {
 ?X a :HighTrafficNode .
 ?Y a :HighTrafficNode .
 ?X :wired ?Y .
}
```

Default Logic front end DReW also supports terminological default reasoning described in Section 5.1. We currently only supports ontology in OWL 2 EL.

Example 6.2. *we consider here an access control policy, borrowed from [BFS11] and couched into a terminological default KB $\Delta = \langle L, D\rangle$, where the the TBox of L is*

$$\mathcal{T} = \left\{ \begin{array}{l} Staff \sqsubseteq User, \quad Blacklisted \sqsubseteq Staff, \quad Deny \sqcap Grant \sqsubseteq \bot, \\ UserRequest \equiv \exists hasAction.Action \sqcap \exists hasSubject.User \sqcap \exists hasTarget.Project, \\ StaffRequest \equiv \exists hasAction.Action \sqcap \exists hasSubject.Staff \sqcap \exists hasTarget.Project, \\ BlacklistedStaffRequest \equiv StaffRequest \sqcap \exists hasSubject.Blacklisted \end{array} \right\}$$

and the defaults are

$$D = \left\{ \begin{array}{c} \dfrac{UserRequest(X) : Deny(X)}{Deny(X)}, \\ \dfrac{StaffRequest(X) : \neg BlacklistedStaffRequest(X)}{Grant(X)}, \\ \dfrac{BlacklistedStaffRequest(X) : \top}{Deny(X)} \end{array} \right\}.$$

Informally, D expresses that users normally are denied access to files, staff is normally granted access to files, while to blacklisted staff any access is denied.

6.1.2 Implementation Details

The main part of DReW is written in Java 7, as Java platform is universally available on all the major operation systems and there are many Java libraries for semantic web, e.g., OWL API. The supporting scripts are in bash so that the user can use DReW as a command line tool. The source code is organized by Apache Maven[3], which makes the project building process easier and in particular can handle the library dependencies. The development is version controlled under Git[4] and hosted at GitHub[5].

DReW uses OWL API[6][HB11] for parsing and representing ontologies. OWL API supports all the major OWL syntax, including RDF/XML, OWL/XML, and Turtle formats. The parser for dl-rules is generated by JavaCC[7]. The input format of dl-rules follows the undocumented syntax used in dl-plugin v1.7[8] of dlvhex, which is an extension of DLV syntax. This format is very close to the mathematical notion used in this thesis. For CQ front end, we use Jena API[9] [Car+04] to parse SPARQL queries.

The underlying Datalog engine we use is DLV[10] [Leo+06], which supports both answer set semantics and well-founded semantics. DLV itself is a command line tool implemented in C++, but there is a Java wrapper (DLV-Wrapper[11] [Ric03]) available.

Supported Operation Systems The DReW system is not designed for a particular operation system (OS). The Java platform is available on all the major OSes. The binary of DLV is available on Linux, FreeBSD, Windows, and Mac OS X (Intel PC). Therefore DReW should be able to run on all these OSes. We have tested it on Ubuntu Linux 12.10, Windows 7, and Mac OS X 10.8.

[3] http://maven.apache.org/
[4] http://git-scm.com/
[5] https://github.com/ghxiao/drew
[6] http://owlapi.sourceforge.net/
[7] https://javacc.java.net/
[8] http://www.kr.tuwien.ac.at/research/systems/dlvhex/dlplugin.html
[9] http://jena.apache.org/
[10] http://www.dlvsystem.com/dlv/
[11] http://www.dlvsystem.com/dlvwrapper/

6.2 Use Cases

The DReW system is already used in several practical applications. In this section, we present two use cases of DReW system in the EDImine project[12], which is about inter-organizational business process mining [Eng+11] in the context of Electronic Data Interchange (EDI) [KB96]. EDImine project uses OWL ontology to model and process the EDI messages and other Business information (BI). The DReW system is used to perform rule based reasoning over the ontologies.

The first use case is to analyze Success Factor (SF), and the second one is to analyze Key Performance Indicator (KPI). Two use cases need different reasoning services from DReW.

6.2.1 Success factor for Inter-organizational relationships

In the business domain, success factor (SF) is an element that influences a mission of an organization or project [Dan61; Roc79]. For example, a success factor which influences *Performance* is *Trust* [CF05; LLL09]. Understanding success factors in inter-organizational relationships (IORs) helps organizations managing their collaboration toward right direction. In this use case, we conducted a review of 182 publications published during 2000–2012 identifying SFs related to IORs as well as study their influencing relationships on each other. Based on the manually extracted influence relations, we want to derive implicit relations and understanding the "core" of these relations. Specifically, we are using DReW for the two tasks:

(1) computing all the implicit influencing relations, and

(2) retrieving a relative part fragment of the extracted influencing relations which is complete w.r.t the original inputs. This fragment is useful for the understanding.

Modeling Success Factors and Influencing Relations

Success Factor Ontology We build an $\mathcal{EL}$ ontology for success factor. The 56 success factors with a hierarchy structure, manually extracted from the literature reviews, are naturally modeled as an OWL ontology O_{SF}, in which constructs are modeled as OWL concepts. The constructs having the same definition or the same measurement are referred as the same SFs. For example, the construct `Collaboration`, `Cooperation`, `Integration` have similar definitions which relate to co-working among business partners, or in DL syntax:

$$\texttt{Collaboration} \equiv \texttt{Cooperation} \equiv \texttt{Integration}$$

Some constructs have hierarchy relations. For instance, by considering the definition of construct *Connectedness* described in [Che11] as "Connectedness indicates the dependence on each other for assistance, information, commitments or in respect of other behaviors that encourage coordination among individuals, departments or organizations", we conclude that `Connectedness` is a part of `Dependency` since according to its definition it is a kind of dependency in terms of behaviors or relationships, which is in turn modeled as a subclass between

[12] http://edimine.ec.tuwien.ac.at/

concepts:

$$\texttt{Connectedness} \sqsubseteq \texttt{Dependency}.$$

Influencing relationship According to the review, 263 influencing relations between the constructs are found. The relationships are modeled as Datalog style facts. For instance, the relation "Trust influences Performance" is model as a fact $inf(\texttt{Trust}, \texttt{Performance})$. Note that here we follow the punning of concepts and individuals as in OWL 2[13], and constructs `Trust` and `Adaptability` are treated as individuals.

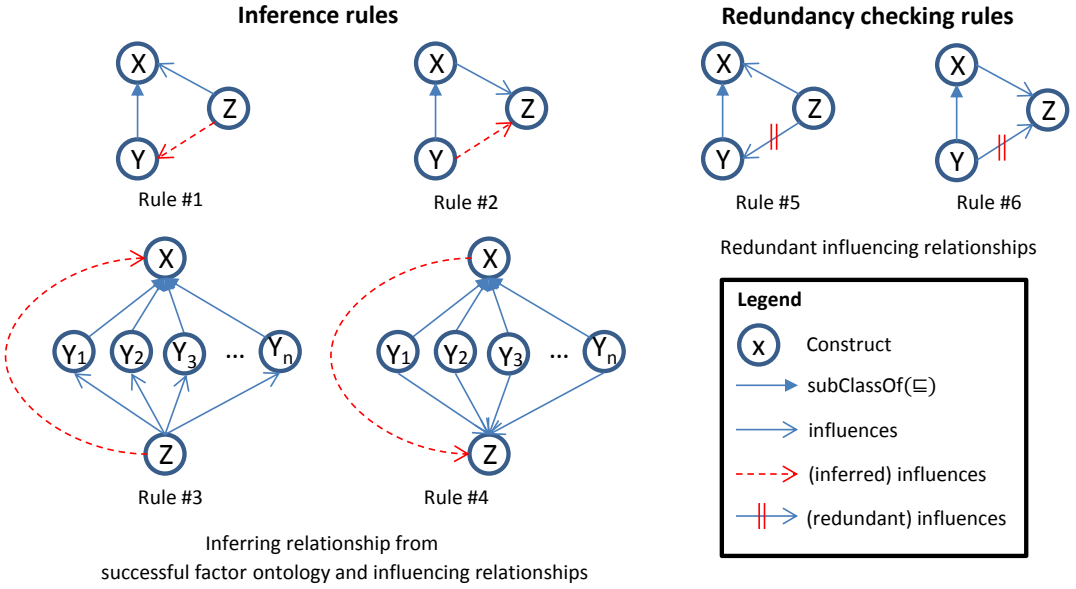


Figure 6.2: Inference rules and redundancy checking rules

Rules for Influencing relationship inference and Redundancy checking

In this step, we model the inferencing of influencing relationship and redundancy checking using rules.

Influencing relationship inference rules New influencing relationships which are not explicitly found in the studies are inferred by inference rules which consider the successful factor ontology and influencing relationships.

We develop four inference rules as depicted in Figure 6.2 (Rule #1, #2, #3, and #4). For instance, rule #1 is says if construct Z influences X, then Z influences all the subclasses of X (e.g. Y). Rule #3 says that if construct Z influences all the subclasses $(Y_1, \ldots, Y_n)$ of X and $X \equiv Y_1 \sqcup \ldots \sqcup Y_n$, then Z influences X. We can express them using the Datalog syntax mixed with DL axioms:

[13] http://www.w3.org/2007/OWL/wiki/Punning

$$inf(Z, Y) \leftarrow inf(Z, X), Y \sqsubseteq X \tag{6.1}$$
$$inf(Y, Z) \leftarrow inf(X, Z), Y \sqsubseteq X \tag{6.2}$$
$$inf(Z, X) \leftarrow inf(Z, Y_1), \ldots, inf(Z, Y_n), X \equiv Y_1 \sqcup \ldots \sqcup Y_n \tag{6.3}$$
$$inf(X, Z) \leftarrow inf(Y_1, Z), \ldots, inf(Y_n, Z), X \equiv Y_1 \sqcup \ldots \sqcup Y_n \tag{6.4}$$

Again, in these rules, we used the OWL 2 punning of individuals and concepts; X is treated as a variable for individual in $inf(Z, X)$ and as a variable for an OWL concept in $Y \sqsubseteq X$.

Redundancy inference checking rules To better understanding the influencing relationships, we can extract a relatively small fragment by identifying redundant relationships. Two examples of the redundancy checking rules are depicted in rule #5 and #6 of Figure 6.2. For example, rule #5 said that if Z influences X and Y subclass of X, then influencing relation between Z and Y is redundant. and formally as as follows:

$$redundant\_inf(Z, Y) \leftarrow inf(Z, X), Y \sqsubseteq X, inf(Z, Y) \tag{6.5}$$
$$redundant\_inf(Y, Z) \leftarrow inf(X, Z), Y \sqsubseteq X, inf(Y, Z) \tag{6.6}$$

Reasoning using DReW

Note that there are variables in Rule (6.1) to (6.6) for reasoning of subclass relations. In order to apply these rules over O_{SF}, we slightly abuse of the DReW system to do some simple yet useful high order reasoning:

(1) Using DReW system, we apply the Datalog rules of materialization calculus K_{sc} [Krö10; Krö11], which is an extension of the Datalog rewriting for instance query of $\mathcal{SROEL}(\sqcap, \times)$ ontologies (see section 4.1), for classifying the ontology O_{SF}. By adding one additional rule

$$\rho : subClass(A, B) \leftarrow inst\_sc(A, B, A)$$

we have that $K_{sc}(L) \cup \{\rho\} \models subClass(A, B)$ iff $L \models A \sqsubseteq B$ for any $\mathcal{SROEL}(\sqcap, \times)$ ontologies L. Note that the ontology O_{SF} uses some disjunctions which are not in $\mathcal{SROEL}(\sqcap, \times)$, so the results might be incomplete. We have to fix the incompleteness by manually adding some missing $subClass$ facts.

(2) Keep the influence facts as Datalog facts.

(3) Replace all the $X \sqsubseteq Y$ by $subset(X, Y)$ in the inference rules and redundancy checking rules. For instance, rule 6.1 is transformed to

$$inf(Z, Y) \leftarrow inf(Z, X), subClass(Y, X).$$

(4) Put the above rules together into a single program and use DLV to compute the model and filter out the facts with predicates $redundant\_inf$.

We find that half of the inferences directly from the literature reviews are redundant. These "core" influencing relations can be further analyzed by business experts.

6.2.2 Key Performance Indicator

A performance indicator or key performance indicator (KPI) is a performance measure indicating the success of particular activities or objectives [Par10]. The EDImine project aims at deriving knowledge by means of business performance from Electronic Data Interchange (EDI) messages exchanged between business partners. The objective is to develop business cockpit supporting business performance analysis based on the Balanced Scorecard (BSC) methodology [Kra+12]. The essence of BSC is to align business objectives with Key Performance Indicators (KPIs) via success factors [KN92]. That is success factors are used as intermediary elements connecting business objectives to KPIs. Therefore there are three main elements (i.e. business objectives, success factors, and KPIs) playing key roles in the BSC implementation. We use DReW system for (i) deriving KPIs from EDI messages as well as (ii) connecting those elements together.

The EDI ontologies are developed for capturing business information in EDI messages at a semantic level according to EDIFACT standard [Ber94]. The ontologies used in this work are developed in OWL and has several components. Figure 6.3 describes the overview of ontologies architecture.The EDIFACT standard as well as the specification of EDIFACT message types are described in the EDIFACT Standards Ontology, EDIFACT Message Ontology, and Message Types KB. The values in EDIFACT messages are parsed into the ontologies according to the aforementioned ontological model. For more details we refer the reader to [Eng+12]. Furthermore, in order to represent those parsed values at the conceptual level which is understandable

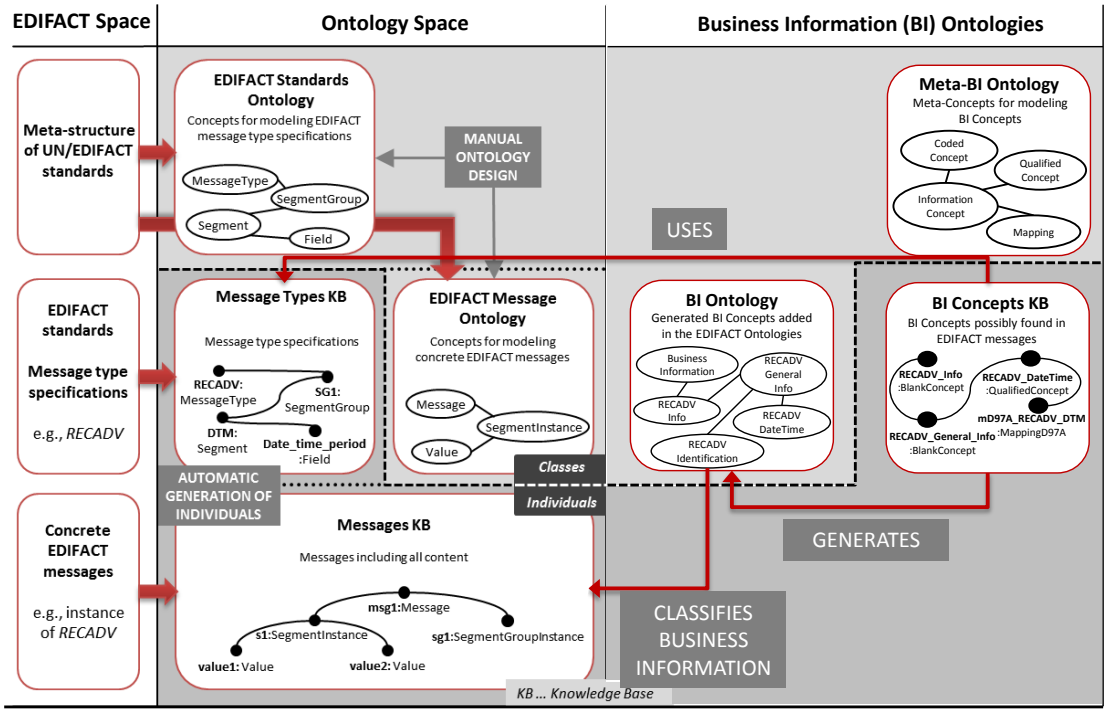


Figure 6.3: EDIFACT ontologies and business information ontologies

for an analysis task we develop business information ontologies to group those values into business information concepts. The Meta-BI Ontology and BI Concept KB describes the mapping between business information concept. By using the mapping information, the BI Ontology is generated on top of EDIFACT ontologies. The BI Ontology described high-level business information concept such as ordered quantity, invoiced amount, order number, etc. Such business information concepts are used to classify EDIFACT values into the corresponding business information. The detail of the BI Ontology can be found in [Kra+12].

The EDIFACT and Business Information ontologies are further used as an information infrastructure for implementing BSC as illustrated in Figure 6.4. As mentioned earlier, the objective of this work is to develop business analysis cockpit which allows users implement the BSC to evaluate their business objectives at strategic level. From this, the user can define their own business objectives and select the relevant success factors that suit with their objectives. By providing knowledge base of success factors and the related KPIs which in turn link to the raw EDIFACT data in the ontologies, the system can infer the relationship between business objectives via the success factors as well as suggest related KPIs which is possible to calculated from the available data. The implementation of the BSC and the reasoning task is support by the DReW system. The advantage of using DReW system is twofold. (1) Comparing to general DL reasoners (Pellet, Hermit, etc.), the DReW system performs much faster when dealing with large ABoxes. (2) It allows us working in combination with OWL and logical rules. While the main ontologies are modeled in OWL, some of inference rules can be model as datalog style rules.

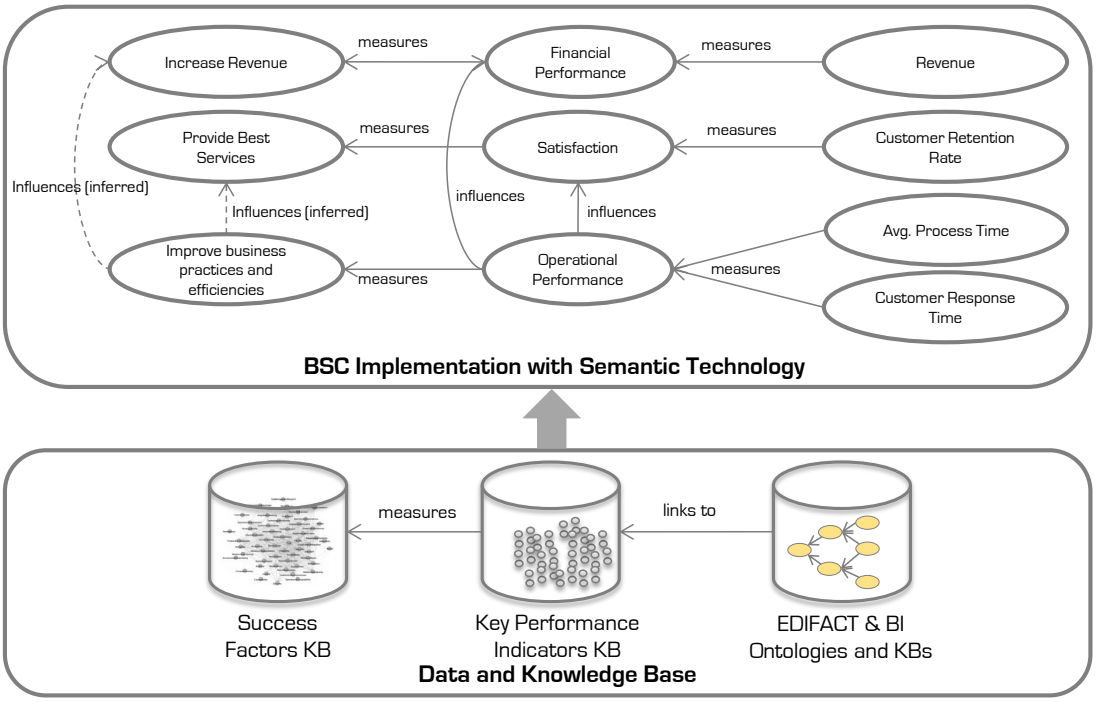


Figure 6.4: The semantic implementation of balanced scorecard

Find business objective which has no KPIs as its measurement
businessobjectiveWithKPI(X):- DL[measures](Y, X), DL[BusinessObjective](X), DL[KPI](Y). businessobjectiveWithNoKPI(X):- DL[BusinessObjective](X), not businessobjectiveWithKPI(X).
Check if the BSC model doesn't contain any business objectives
notexistBusinessObjective:- not existBusinessObjective. existBusinessObjective:- DL[BusinessObjective](X).
Check if a business objective have total KPI weight as 100
q1(O, K, W):- DL[measures](K, O), DL[hasWeight](K, W). sum(O, X):- X = #sum{W, K : q1(O, K, W)}, DL[BusinessObjective](O). BO_WeightViolation(O):- sum(O, X), X != 100. safeBusinessObjective(O):- DL[BusinessObjective](O), not BO_WeightViolation(O).

Table 6.1: Rules in KPI use case

Modeling rules in Datalog syntax provides us more intuitive way to formulate the complex rules.

We show one example of using dl-programs for checking the completeness of BSC model. The complete BSC model should contains some business objectives; each business objectives must be measured by some KPIs and the total weight of KPIs of each business objective must be sum as 100. These rules are modeled in dl-programs as shown in Table 6.1 in order to check if there are some violation in the BSC model.

6.3 Related Systems

NLP-DL NLP-DL [Eit+04a; Eit+08a] is the first experimental system for dl-programs which supports both well-founded semantics and answer set semantics. This prototype is implemented in scripting language PHP and is mainly used for demonstration. NLP-DL is superseded by DLVHEX.

DLVHEX The most related system of DReW is DLVHEX, the successor of NLP-DL, targeting HEX-programs. We list some major differences in term of features as below.

- **External sources.** DLVHEX is a reasoning engine for HEX-programs, which are an extension of dl-programs towards integration of external computation sources (not necessarily DL ontologies). DLVHEX has plugins for several external resources, e.g. action, merging and string.
- **Expressivity of the DL component.** DLVHEX supports DL ontologies by dl-plugins using the external reasoner RacerPro. The DLs supported by DReW depends on the underlying rewriting algorithms. Current implementation of DReW support OWL 2 RL and OWL 2

EL; the algorithm for Horn-$\mathcal{SHIQ}$ is ready to implement. In contrast, DLVHEX supports more expressive DL ($\mathcal{SHIQ}$) via RacerPro.

- **Semantics of dl-programs.** For dl-programs, DLVHEX supports only answer set semantics, while DReW supports both answer set semantics and well-founded semantics.

- **Safeness conditions.** DLVHEX explicitly requires the DL-safeness condition that every variable occurs in the rules must occur in some non-DL body atoms. For instance, the rule $q(X) \leftarrow DL[Q](X)$ is not DL-safe, because the variable X does not occur in non-DL atoms. In practice, we usually add auxiliary atoms to the rule to ensure the DL-safeness, e.g. $q(X) \leftarrow DL[Q](X), dom(X)$, and explicitly insert assertions $dom(a)$, for all the individuals a that occur in the ontology and rules. This step of retaining DL-safeness is domain dependent and tedious and makes the rules verbose. In DReW, we implicitly apply the DL-safeness condition, and allow rules like $q(X) \leftarrow DL[Q](X)$.

- **Datatypes.** Simple XML Schema Datatypes can be used in RDF and OWL. Current implementation of DLVHEX does not support any datatypes, while DReW supports some simple datatypes including string (e.g., `"abc"^^xsd:string`) and integer (e.g., `"123"^^xsd:integer`).

MOR MOR is an experimental prototype system for first order rewritable dl-programs [Sch10]. MOR supports DL-Lite ontologies and acyclic dl-rules. The backend engine is RDBMS (currently PostgreSQL) rather than ASP engine. Compared with other systems for dl-programs, MOR is designed to work with larger data set, but sacrificing expressivity.

ASPIDE ASPIDE[14] is a integrated development environment for ASP [FRR11], which has a list of input, rewriting, and output plug-ins for ASPIDE [15][Feb+12; Nar+13]. Some notable plugins are:

- *ASPIDE OWL to Facts* is a rewriting plugin which creates a Datalog view in term of facts representing the ABox of an ontology;

- *ASPIDE OWL to DLVEx* is a rewriting plugin which creates a Datalog Exist program out of an OWL ontology (supports DL-Lite only);

- *ASPIDE Requiem* is a rewriting plugin exploiting the Requiem query rewriter [PMH09] for query answering on lightweight ontologies.

These plugins let ASPIDE access DL ontologies. Compared with dl-programs, ASPIDE can not express the bidirectional info between the ontology and rules.

[14] https://www.mat.unical.it/ricca/aspide/

[15] https://www.mat.unical.it/ricca/aspide/plugins.html

CHAPTER 7

Performance Evaluation

In this chapter, we present the performance evaluation of inline evaluation by DReW system comparing existing systems. The results imply that inline evaluation outperforms the classical approaches of reasoning over dl-programs. The benchmarks suites also show that dl-programs are a expressive and powerful language for query answering over DL ontologies.

7.1 Introduction

We build five benchmark suites (namely Graph, University, GeoData, EDI and Policy) in different scenarios for evaluating the following aspects:

- Effects of reduction of DL calls. The main motivation of this thesis is to reduce the external calls to DL reasoners. Two suites (Graph and Policy Suite) are in particular used to show comparison results when multiple calls to DL reasoners are required in the classical approach.
- Scalability. We are interested in the performance of the DReW system over dl-programs with large ABoxes. We generate relatively large ABoxes in the GeoData, University and EDI benchmark suites.
- Expressivity. Since dl-programs are a very expressive language, we build dl-programs with features (e.g., non-monotonic negations, multiple rules, recursions, and dl-inputs) which are not commonly available in other query languages.
- Underlying DL rewriting algorithms. Recall that we implemented rewriting of two styles: direct rewriting (i.e. RL rewriting) and reification based rewriting (i.e. EL rewriting) in DReW system. We evaluate benchmarks using both settings to compare the performance.

We run DReW and DLVHEX systems on benchmark suites and have the following insights:

- DReW shows a clear advantage when multiple calls to the external DL reasoners are required by classical dl-program reasoners.
- The performance of DReW on large ABoxes scales polynomially in general.
- Queries with more expressive features do not affect much on the performance of DReW.
- In most of the evaluations, the direct rewriting approach (RL) is faster than the reification-based rewriting (EL).

7.2 Experiments

We present the details of experiments in this section. Firstly, in Section 7.2.1 we introduce five benchmark scenarios used in the evaluation. Next we show the experimental platform in Section 7.2.2. Lastly, the detailed results of the evaluation on theses benchmark suites are reported in Section 7.2.3.

7.2.1 Benchmark scenarios

In our setting, a *benchmark instance* is a pair of an ontology and a dl-program. A *benchmark suite* consists of a set of ontologies sharing the same TBox but with different ABoxes, and a set of dl-programs. Thus every combination of an ontology and a dl-program from a benchmark suite forms a benchmark instance. We evaluate all the instances from the all benchmark suites.

We build five benchmark suites in this chapter by adapting exiting benchmarks and by creating new ones from other domains. A summary of the ontology component of these benchmarks suites are shown in Table 7.1. The Graph and Policy benchmarks are syntactical benchmarks modifying existing benchmarks. We had a problem when evaluate the scalability on ontologies with relatively large ABoxes, since such ontologies in the DL community are not common. In contrast most of the benchmarks used in the DL community are either with small ABoxes or only for TBox reasoning (e.g., concept classification and query rewriting for DL-Lite)[1]. The LUBM test suite is the only one for ABox benchmarking which we are aware of. To solve this, we collaborated with other research projects and developed two benchmark suites: the Geo-Data benchmark suite is developed with MyITS project about the geography domain; the EDI benchmark suite is developed with EDIMine project about the business domain.

(1) Graph Benchmark Suite The graph benchmark suite is adapted from that in [Eit+04a]. For some instances in this benchmark suite, classical evaluation method needs multiple calls to the DL reasoner. So it is particularly useful for evaluating the effect of inline evaluation of dl-programs reducing many calls to external DL reasoners.

The graph ontology uses an empty TBox. The individuals in the ontology model the nodes of the graph; the property arc is used for the arcs between the nodes. The task in this scenario is

[1]The problem of lacking reasonable benchmark suites is confirmed by communication with several professors in the DL community

Suite Name	DL	#CON	#OP	#DP	#TA	#RA
Graph	$\mathcal{AL}$	1	1	0	0	0
University	$\mathcal{ALEHI}$	128	28	7	183	6
GeoData	DL-Lite	345	32	5	397	19
EDI	$\mathcal{SHOI}$	152	61	35	303	68
Policy	$\mathcal{EL}$	9	3	0	6	0

Table 7.1: TBox and RBox size of benchmark suites. #CON=number of concepts, #OP=number of object properties, #DP=number of data properties, #TA=number of TBox axioms, #RA=number of RBox axioms

to compute the transitive closure of the graph in the ABox by dl-programs. Different programs may need different numbers of calls to DL reasoners in classical approach.

(2) University Benchmark Suite Lehigh University Benchmark (LUBM) is a widely used benchmark in the DL community, which consists of a TBox, an ABox generator and a set of predefined queries [GPH04]. ModLUBM is a modified DL-Lite version of LUBM [Lut+12]. The concepts and properties used in ModLUBM is a superset of that in LUBM. Compared with the original LUBM data generator, ModLUBM additionally introduces customizable number of subclasses of Department, Student and Professors by adding concepts Subj*i*Department, Subj*i*Student and Subj*i*Professors. Therefore, the results of ModLUBM ABox generator can be much larger than those of the original LUBM generator. The TBox of ModLUBM is restricted to DL-Lite Axioms.

To make the evaluation more interesting, we combined the TBoxes and RBoxes from LUBM and ModLUBM into our University benchmark suite. The resulting ontologies are classified as $\mathcal{ALEHI}$. For the ABox part, we use the generator of the ModLUBM.

(3) GeoData Benchmark Suite OpenStreetMap (OSM) is a well known user-generated street map [HW08]. OSM has huge amount data which uses a topological data structure with nodes, ways, relations and tags. GeoData Benchmark suite is a novel benchmark derived from OSM. This ontology of this benchmark suite is developed in KBS group of Vienna University of Technology in the context MyITS project[2].

The TBox used in GeoData benchmark suite is GeoConceptsMyITS-v0.9-Lite[3], which is a DL-Lite ontology. The ABoxes are extracted from OSM. We first load the OSM raw data to PostgreSQL database with PostGIS extension using osm2pgsql[4]. Then the features (e.g., leisure, shop, metro, bus, tram, and admin) are extracted from PostGIS. The spatial relations (e.g., next, within) between the features are computed and materialized into OWL files by our Python scripts (using GDAL library).

[2] http://www.kr.tuwien.ac.at/research/projects/myits/

[3] http://www.kr.tuwien.ac.at/staff/patrik/GeoConceptsMyITS-v0.9-Lite.owl

[4] http://wiki.openstreetmap.org/wiki/Osm2pgsql

(4) EDI (EDIMine) Benchmark Suite Electronic data interchange (EDI) is a approach of transferring data between different computer systems or computer networks [KB96].

The EDI Benchmark suite is developed in Institute of Software Technology & Interactive Systems of Vienna University of Technology in the context of EDIMine Project[5]. This benchmark scenario is a part of the KPI use case in Section 6.2.2.

The TBox component of the benchmark suite contains the EDI ontology (derived from the UN/EDIFACT standard [Ber94]) and the business information (BI) ontology. The ABoxes are converted from the EDI messages. The dl-programs are used for querying messages and the results can be used for further analysis by business experts.

(5) Policy Benchmark Suite This benchmark suite is used for evaluating the default logic front end of DReW. we consider here an access control policy terminological default KB $\Delta = \langle L, D\rangle$ as in Example 6.2, where the the TBox of L and the defaults D are shown bellow:

$$\mathcal{T} = \left\{ \begin{array}{l} \mathit{Staff} \sqsubseteq \mathit{User}, \quad \mathit{Blacklisted} \sqsubseteq \mathit{Staff}, \quad \mathit{Deny} \sqcap \mathit{Grant} \sqsubseteq \bot, \\ \mathit{UserRequest} \equiv \exists \mathit{hasAction}.\mathit{Action} \sqcap \exists \mathit{hasSubject}.\mathit{User} \sqcap \exists \mathit{hasTarget}.\mathit{Project}, \\ \mathit{StaffRequest} \equiv \exists \mathit{hasAction}.\mathit{Action} \sqcap \exists \mathit{hasSubject}.\mathit{Staff} \sqcap \exists \mathit{hasTarget}.\mathit{Project}, \\ \mathit{BlacklistedStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubject}.\mathit{Blacklisted} \end{array} \right\}$$

$$D = \left\{ \begin{array}{l} \mathit{UserRequest}(X) : \mathit{Deny}(X) / \mathit{Deny}(X), \\ \mathit{StaffRequest}(X) : \neg \mathit{BlacklistedStaffRequest}(X) / \mathit{Grant}(X), \\ \mathit{BlacklistedStaffRequest}(X) : \top / \mathit{Deny}(X) \end{array} \right\}$$

Informally, D expresses that users normally are denied access to files, staff is normally granted access to files, while to blacklisted staff any access is denied. We generate ABox assertions about the users and the requests and use DReW and DLVHEX to query whether the requests are granted or denied.

7.2.2 Platform

We mentioned in section 6.3, that we are aware of three existing systems for dl-programs, namely NLP-DL, DLVHEX, and MOR. However, we only compared DReW with DLVHEX for the following reasons.

- NLP-DL is implemented in PHP and only available from a web interface. It is not maintained and there is no easy way to automate the evaluation. Most importantly, it is superseded by DLVHEX.
- MOR is an experimental prototype system supports only DL-Lite ontologies and first-order rewritable dl-programs. Instead of Datalog engine in other dl-program reasoners, MOR uses a relational database as backend. Setting up such database is non-trivial and not well-documented.

[5] http://edimine.ec.tuwien.ac.at/

The evaluation is performed on a Linux Server running Ubuntu 12.04. The DLV used in DLVHEX and DReW is DLV version 2012-12-17 and the Java version is Oracle jdk1.7.0_21. we use DLVHEX 1.7.2 as the current version 2 does not support the old input formats of dl-programs (which is the same as DReW) and compared to version 1.7.2, there is no much performance difference on the dl-programs used in this thesis. For the dl-plugin of DLVHEX, we use RacerPro version 1.9.2 beta (released on October 25, 2007). Unfortunately, we can not test against the latest version of RacerPro 2.0 preview because it is not fully compatible with DLVHEX. For DReW system, we use DReW[RL] (resp. DReW[EL]) to denote using the RL (resp. EL) rewritings.

To make the evaluation fully automated, we implemented an experiment platform using the HTCondor system[6] [TTL05]. The platform is inspired by the VCWC system [Cha+13] used for the ASP competition 2013. Thanks to HTCondor, we can schedule the experiments and restrict the resources of each test. The workflow of the evaluation is as following:

(1) Benchmark suite generation. Each benchmark suite is physically put in one folder with two subfolders: (a) ontologies (b) programs. Then we can evaluate all the instances from all the combination of the ontologies and programs.

(2) HTCondor submission files generation. We write scripts to generate the submission files for HTCondor, where we specify that each run of DReW system is restricted to 8G of memory (6G allocated for Java VM). Since each run of DReW creates a new process of DLV, multiple runs of DReW can be performed simultaneously. In case of DLVHEX, we have to run them sequentially because DLVHEX communicates RacerPro through fixed ports (8080 and 8088).

(3) Execution of the benchmarks. We submit the HTCondor submissions files, and HTCondor takes care of the execution. The results are stored in log files.

(4) Log files analysis. We run scripts for analyzing the log files and there are three possible results for each run:

   (a) the reasoner finished in time and successfully output results;

   (b) the reasoner failed on the instance, which is normally caused by out of memory; and

   (c) the reasoner run out of time (> 600s).

7.2.3 Evaluation

We present the detailed evaluation results of DReW and DLVHEX on the five benchmark suites.

(1) Graph Benchmark Suite

To build arbitrary large ABoxes, we did not directly adopt the instances of the graph from [Eit+04a]. Instead, the ABoxes are generated by a random graph generator[7], which generates random simple connected graphs with prescribed degree sequence [VL05]. We generated ten ABoxes of graphs (graph-n) with different numbers of nodes and edges, which are listed in Table 7.2. We

[6] http://research.cs.wisc.edu/htcondor/

[7] http://www-rp.lip6.fr/~latapy/FV/generation.html

note that here all the concept assertions are actually declaration of the nodes and all the object property assertions are of the form $arc(a, b)$ for edges (a, b) in the graph. The numbers of edges are always set to five times of the number of the nodes, that is, larger graphs are more sparse. The OWL files of these ABoxes are very small when measured in MB on the disk.

To compute the transitive closure of the arc relations in the ABox, we use four dl-programs in Table 7.3, which are evaluated by DReW[RL], DReW[EL] and DLVHEX. The evaluation results are shown in Table 7.4. We make the following observations:

- DLVHEX needs different numbers of calls to RacerPro for different programs. By analyzing the log files, we found that DLVHEX access only once to RacerPro for tc_1 and tc_2, but multiple times for tc_3 and tc_4. This could explain that for DLVHEX the performances on tc_1 and tc_2 are better than those on tc_3 and tc_4.

- For t_3 and t_4, DLVHEX fails on graphs with more than 300 nodes because that RacerPro crashes and reports an out of memory problem. The reason could be that RacerPro is a 32bit program, which can only access up to 4G of memory. Every time DLVHEX queries RacerPro with a dl-input, RacerPro will actually create a copy of the whole ontology. When there are many such queries, RacerPro may run out of memory because of these copies.

- DReW is always faster than DLVHEX. When more than one calls to RacerPro is needed (tc_3 and tc_4), the advantage is more clear. This shows that inline evaluation strategy indeed reduces the overhead of calling external DL reasoners.

- For all of the three reasoners, the runnings on tc_2 are in quadratic time w.r.t the number of nodes, but the runnings on the rest programs are roughly in exponential time. We checked the log of DReW[RL] and DReW[EL] and found that most of the running time (over 99%) is spent on DLV. It shows that DLV performs better on the linear recursion version (used in tc_2) of this transitive closure computation.

- Comparing the performance of DReW[EL] and DReW[RL], we found that DReW[EL] is in general faster than DReW[RL].

To conclude, in graph benchmark suite DReW outperforms DLVHEX clearly, especially when multiple calls to RacerPro are needed by DLVHEX. The failures of the runs of DLVHEX can come from the either the DLV part or the RacerPro part. DReW[EL] runs faster than DReW[RL] on most instances.

Ontology	#IND	#CA	#OPA	#DPA	File size
graph-10	10	10	50	0	0M
graph-100	100	100	500	0	0M
graph-200	200	200	1000	0	0M
graph-300	300	300	1500	0	0M
graph-400	400	400	2000	0	0M
graph-500	500	500	2500	0	0M
graph-600	600	600	3000	0	0M
graph-700	700	700	3500	0	0M
graph-800	800	800	4000	0	0M
graph-900	900	900	4500	0	0M

Table 7.2: ABox Sizes of the Graph Benchmark Suite. #IND: individuals, #CA: concept assertions, #OPA: object property assertions, #DPA: data property assertions.

tc_1	*This program extracts the arc relations from the ontology and computes the closure by binary recursion* tc(X, Y) :- DL[arc](X, Y). tc(X, Y) :- tc(X, Z), tc(Z, Y)
tc_2	*This program extracts the arc relations from the ontology and computes the closure by linear recursion* edge(X, Y) :- DL[arc](X, Y). tc(X, Y) :- edge(X, Y). tc(X, Y) :- edge(X, Z), tc(Z, Y).
tc_3	*This program extracts the arc relations from the ontology and computes the closure by recursion while feeding back the arc relations* tc(X, Y) :- DL[arc](X, Y). tc(X, Y) :- DL[arc$\uplus$ tc; arc](X, Z), tc(Z, Y).
tc_4	*This program extracts the arc relations from the ontology while feeding back the arc relations by tc and computes the closure by recursion* edge(X, Y) :- DL[arc $\uplus$ tc; arc](X, Y). tc(X,Y) :- edge(X, Y). tc(X, Y) :- DL[arc](X, Z), edge(Z, Y).

Table 7.3: dl-programs in the Graph benchmark suite

	tc1	tc2	tc3	tc4
graph-10	1.1	1.0	1.1	1.2
graph-100	2.6	1.5	2.5	2.5
graph-200	12.1	2.2	11.8	10.7
graph-300	40.4	3.0	38.7	38.8
graph-400	97.8	4.3	93.4	90.4
graph-500	195.3	5.8	192.3	194.6
graph-600	350.5	7.9	350.3	365.4
graph-700	579.87	10.3	562.2	577.1
graph-800	OT	12.3	OT	OT
graph-900	OT	16.7	OT	OT

(a) Results by DReW[RL]

	tc1	tc2	tc3	tc4
graph-10	1.0	1.1	1.1	1.0
graph-100	2.5	1.4	2.5	2.4
graph-200	11.3	2.2	10.5	11.1
graph-300	40.1	3.1	32.5	31.9
graph-400	93.6	4.3	75.8	82.8
graph-500	185.0	5.8	163.9	153.6
graph-600	359.6	7.9	259.3	263.2
graph-700	556.1	9.9	411.1	430.2
graph-800	OT	12.9	525.2	555.2
graph-900	OT	16.2	OT	OT

(b) Results by DReW[EL]

	tc1	tc2	tc3	tc4
graph-10	1.4	0.3	1.8	1.7
graph-100	2.8	0.5	9.8	10.4
graph-200	13.6	1.3	52.8	57.5
graph-300	52.5	2.6	162.6	160.4
graph-400	133.6	4.6	OM	OM
graph-500	266.1	6.9	OM	OM
graph-600	478.6	10.1	OM	OM
graph-700	OT	13.7	OM	OM
graph-800	OT	18.4	OM	OM
graph-900	OT	23.9	OM	OM

(c) Results by DLVHEX

Table 7.4: Evaluation results of Graph benchmark suite (OT: > 600s; OM: out of memory)

(2) University Benchmark Suite

P_1	*This program queries all the students X and universities Y, such that X gets both undergraduate degree and master degree from Y, but does not get doctoral degree from Y.* q(X, Y) :- DL[undergraduateDegreeFrom](X, Y), DL[mastersDegreeFrom](X, Y), not DL[doctoralDegreeFrom](X, Y).
P_2	*This program queries all the students X and courses Y, such that X is the teaching assistant of Subject2Course Y, but X is not a student in Subject2.* q(X, Y) :- DL[teachingAssistantOf](X, Y), DL[Subj2Course](Y), DL[Student](X), not DL[Subj2Student](X).
P_3	*This program queries all the people who has degrees from two different places.* q(X, Y1, Y2) :- DL[degreeFrom](X, Y1), DL[degreeFrom](X, Y2), Y1 != Y2.
P_4	*This program queries all the triples (X, Y, Z) such that X is the advisor of Y, and X and Y are coauthors of P.* coauthor(P, X, Y) :- DL[publicationAuthor](P, X), DL[publicationAuthor](P, Y), X != Y. q(P, X, Y) :- coauthor(P, X, Y), DL[advisor](X, Y).
P_5	*This program queries all the Persons X who are in the transitive closure of the coauthor relationship of FullProfessor1 in Department0 of University0.* coauthor(X, Y) :- DL[publicationAuthor](P, X), DL[publicationAuthor](P, Y), X != Y. tc_coauthor(X, Y) :- coauthor(X, Y). tc_coauthor(X, Y) :- coauthor(X, Z), tc_coauthor(Z, Y). q(X) :- tc_coauthor(X, "<http://www.Department0.University0.edu/FullProfessor1>").
P_6	*This program queries all the professors who are not in the transitive closure of the coauthor relation of FullProfessor1 in Department0 of University0.* coauthor(X, Y) :- DL[publicationAuthor](P, X), DL[publicationAuthor](P, Y), X != Y. tc_coauthor(X, Y) :- coauthor(X, Y). tc_coauthor(X, Y) :- coauthor(X, Z), tc_coauthor(Z, Y). q(X) :- DL[Professor](X), not tc_coauthor(X,"<http://www.Department0.University0.edu/FullProfessor1>").

Table 7.5: dl-programs in the University benchmark suite

We generated large ABoxes using the generator of ModLUBM [Lut+12] to test the scalability. The incompleteness was set to 5%. The ABoxes are in Table 7.6, in which each Ui stands for an ABox with i universities. We build six dl-programs in Table 7.5. We note that these programs can not be be directly expressed by conjunctive queries, since P_1 and P_2 use negations, P_3 and P_4 use inequalities, and P_5 and P_6 use recursions.

The test programs are evaluated on DLVHEX and DReW. Surprisingly, DLVHEX cannot terminate on the programs with U1 in 600s, so we only list the running time of DReW[RL] and DReW[EL] in Table 7.7. We see that both DReW[RL] and DReW[EL] scale well. For most of the instances, DReW[RL] outperforms DReW[EL].

Ontology	#IND	#CA	#OPA	#DPA	File Size
U1	17138	28023	47640	33201	9M
U2	37403	61924	107362	74710	22M
U4	76710	128160	224805	155951	46M
U6	115998	194106	341204	236702	69M
U8	159193	266617	468928	325563	95M
U10	200992	336870	593388	411768	121M
U15	311875	523097	922763	640069	188M
U20	426837	716556	1265316	876971	259M

Table 7.6: ABox sizes of the University benchmark suite. #IND: individuals, #CA: concept assertions, #OPA: object property assertions, #DPA: data property assertions

	p1	p2	p3	p4	p5	p6
U1	11.5	11.0	12.8	11.3	16.7	17.3
U2	18.5	17.0	20.8	19.3	30.4	28.9
U4	25.8	26.1	35.0	32.9	59.4	56.8
U6	36.0	38.0	41.7	42.2	85.9	90.5
U8	47.0	47.5	59.3	49.1	133.6	136.4
U10	81.2	84.3	94.0	90.5	169.6	175.3
U15	97.9	104.2	117.6	101.6	205.2	229.6
U20	150.8	156.2	165.7	158.0	322.6	337.0

(a) Results by DReW[RL]

	p1	p2	p3	p4	p5	p6
U1	21.1	18.4	21.3	22.6	23.2	25.7
U2	28.3	28.4	29.4	29.6	39.9	40.9
U4	54.9	58.0	61.4	57.9	79.4	82.1
U6	142.1	144.6	186.6	183.0	192.0	192.2
U8	247.7	248.5	261.5	246.0	247.8	248.2
U10	295.7	308.7	306.3	315.5	307.5	306.1
U15	504.7	508.9	503.2	502.3	536.2	533.0
U20	OT	OT	OT	OT	OT	OT

(b) Results by DReW[EL]

Table 7.7: Evaluation Results of the University benchmark suite

(3) GeoData Benchmark Suite

	#IND	#CA	#OPA	#DPA	#next	#within	File Size
Salzburg	12971	13037	539	19513	79615	455	11M
Vienna	33405	33531	1303	50520	292985	2610	36M
Austria	150911	151616	5326	222189	893438	6712	133M
Upper Bavaria	70837	71201	2182	106140	414512	3772	55M

Table 7.8: ABox Sizes of the GeoData benchmark suite. #IND: individuals, #CA: concept assertions, #OPA: object property assertions, #DPA: data property assertions, #next and #within: geographical relations next and within.

P_1 *List all the subway stations with at least two restaurants nearby.*

q(XN_1, XN_2, YN) :- DL[Restaurant](X1), DL[featurename](X_1, XN_1),
DL[Restaurant](X2), DL[featurename](X_2, XN_2), $X_1 \neq X_2$,
DL[next](X_1, Y), DL[next](X_2, Y),
DL[SubwayStation](Y), DL[featurename](Y, YN) .

P_2 *List all the restaurants without cuisine information*

q(XN) :- DL[Restaurant](X), DL[featurename](X, XN), not hasCuisine(X).
hasCuisine(X) :- DL[hasCuisine](X, Y).

P_3 *List all the subway stations in the ontology, but not known from rule part.*

q(X, Xname) :- DL[SubwayStation](X), DL[featurename](X, Xname), *not* metro_stop(Xname).
metro_stop(Stop1) :- metro_next(Line, Stop1, Stop2).
metro_stop(Stop2) :- metro_next(Line, Stop1, Stop2).
% facts of the subway lines
metro_next("U1","Reumannplatz" , "Keplerplatz").
metro_next("U1" , "Keplerplatz" , "Suedtiroler Platz").
...
metro_next("U6", "Handelskai", "Neue Donau").
metro_next("U6", "Neue Donau", "Floridsdorf").

Table 7.9: dl-programs in the GeoData Benchmark Suite (P_1 – P_3)

Recall that the data of the ontology of this suite contain two parts: (1) the features directly extracted from OSM stored in PostgreSQL, (2) the geographical relations computed by our scripts using Python . In this thesis, we extracted features (including types of leisure, shop, muse, metro, bus, tram, admin) from the four areas: Salzburg, Vienna, Austria, and Upper Bavaria. Two geographical relations `next` and `within` are extracted. The statistics of ABoxes is in Table 7.8, where the next and within relations are shown in separated columns.

We build six dl-programs in Table 7.9 and 7.10. Some of the programs use the extracted graph of subway lines of Vienna. The evaluation results are in Table 7.11. Note that we did not list the results of DLVHEX because it can not terminate even on $q1$ over Salzburg or Vienna

P_4	*List all the distances (number of stops) of the subway stations to "Karlsplatz"* .
	q(*YN*, N) :- DL[SubwayStation](X), DL[featurename](X, "Karlsplatz"), DL[SubwayStation](Y), DL[featurename](Y, *YN*), stops(*XN*, *YN*, N).
	stops(X, Y, 1) :- metro_next(Line, X, Y).
	stops(X, Y, N1) :- stops(X, Z, N), metro_next(Line, Z, Y), N1 = N + 1, *not* stops_less_than(X, Y, N1), #int(N1).
	stops_less_than(X, Y, N1) :- stops(X, Y, N2), N2 < N1, #int(N1), #int(N2).
	metro_next(Line, Stop1, Stop2) :- metro_next(Line, Stop2, Stop1).
	% and the facts of the subway lines as in P_3
P_5	*List all the Italian restaurants next to a subway station which can be reached from "Karlsplatz" by one change.*
	q(YN, ZN, L1, L2) :- metro_connect_1_change(L1,L2,"Karlsplatz", YN), DL[SubwayStation](Y), DL[featurename](Y, YN), DL[Restaurant](Z), DL[next](Y, Z), DL[featurename](Z, ZN), DL[hasCuisine](Z, "ItalianCuisine").
	metro_next(Line, Stop1, Stop2) :- metro_next(Line, Stop2, Stop1).
	metro_connect_0_change(L, Stop1, Stop2) :- metro_next(L, Stop1, Stop2).
	metro_connect_0_change(L, Stop1, Stop2) :- metro_connect_0_change(L, Stop1, Stop3), metro_connect_0_change(L, Stop3, Stop2).
	metro_connect_1_change(L1, L2, Stop1, Stop2) :- metro_connect_0_change(L1, Stop1, Stop3), metro_connect_0_change(L2, Stop3, Stop2), L1 != L2.
	% and the facts of the subway lines as in P_3
P_6	*Select restaurants next to "Karlsplatz" with preference: ChineseCuisine > AsianCuisine > Other*
	restaurant(X) :- DL[Restaurant](X), DL[next](X,Y), DL[SubwayStation](Y), DL[featurename](Y, "Karlsplatz").
	chinese_restaurant(X) :- restaurant(X), DL[hasCuisine](X, "ChineseCuisine").
	asian_restaurant(X) :- restaurant(X), DL[hasCuisine](X, "AsianCuisine").
	exists_chinese_restaurant :- chinese_restaurant(X), restaurant(X).
	exists_asian_restaurant :- asian_restaurant(X), restaurant(X).
	sel(X) :- chinese_restaurant(X), exists_chinese_restaurant.
	sel(X) :- asian_restaurant(X), *not* exists_chinese_restaurant, exists_asian_restaurant.
	sel(X) :- restaurant(X), *not* exists_asian_restaurant, *not* exists_chinese_asian_restaurant.
	q(XN) :- sel(X), DL[featurename](X, XN).

Table 7.10: dl-programs in the GeoData Benchmark Suite (P_4 – P_6)

in 20 mins. We see that both the running times of DReW[EL] and DReW[RL] scales well, but DReW[RL] outperforms DReW[EL] clearly.

	p1	p2	p3	p4	p5	p6
Vienna	25.81	25.38	25.47	32.15	28.83	23.31
Salzburg	13.84	14.88	13.35	20.11	15.79	10.78
Austria	78.1	74.5	74.0	81.9	82.2	77.9
Upper Bavaria	33.15	33.36	32.27	36.21	31.65	28.9

(a) Evaluation results by DReW[RL]

	p1	p2	p3	p4	p5	p6
Vienna	60.56	58.93	57.92	65.42	63.31	53.89
Salzburg	21.28	23.89	21.15	30.43	23.25	20.85
Austria	172.0	166.9	169.8	171.9	176.2	175.3
Upper Bavaria	89.09	87.05	85.84	87.42	92.82	82.36

(b) Evaluation results by DReW[EL]

Table 7.11: Query Evaluation of the GeoData Benchmark Suite

(4) EDI Benchmark Suite

ABox	#Invoice	#Order	#IND	#CA	#OPA	#DPA	File Size
EDI_5	1	4	19326	19326	41487	54480	12M
EDI_10	2	8	23063	23063	63674	61305	16M
EDI_25	5	20	36085	36085	141009	85090	28M
EDI_50	10	40	57448	57448	268104	124260	50M
EDI_75	12	63	67893	67893	330384	143621	60M
EDI_100	15	85	77542	77542	387863	161435	70M
EDI_125	15	110	81407	81407	411105	168794	73M
EDI_150	20	130	97343	97343	505978	198040	89M
EDI_175	20	155	101679	101679	532055	206289	94M
EDI_200	25	175	117309	117309	625097	234952	109M

Table 7.12: ABox sizes of EDI benchmark suite. #IND: individuals, #CA: concept assertions, #OPA: object property assertions, #DPA: data property assertions.

In this benchmark suite, we considered two types of EDI messages: D01B[8] (invoices) and D96A[9] (orders). The ABoxes are extracted from the EDI messages of these two types. EDI messages are very compact and contain many information. For example, D01B specification defines 53 segment groups and each group can contain two to ten segments. Each segment will be transformed to at least one ABox assertion. The sizes are shown in the Table 7.12. There are

[8] http://www.unece.org/trade/untdid/d01b/trmd/invoic_c.htm
[9] http://www.unece.org/trade/untdid/d96a/trmd/quotes_c.htm

two columns (#Invoice and #Order) for the number of the corresponding invoice messages and order messages.

We build five dl-programs shown in Table 7.13 and evaluate them. The ontologies are in $\mathcal{SHOI}$ and most of the axioms are in RL. Some import axioms use inverse roles and they are not in EL, so we only evaluated this benchmark suite on DReW[RL] and the evaluation results are in Table 7.14. Again, we did not list the results of DLVHEX because it can not terminate in a reasonable time. In contrast, DReW[RL] behaves almost linearly.

P_1 *get line item invoiced quantities, where the itemnumber = "xxxxxxxxxxxx" (for specific product)*

```
q(Y, N, X, Q) :- DL[INVOIC_LineItemQuantity_invoicedquantity](X),
                 DL[D01B:containedInSegmentInstance](X, S),
                 DL[D01B:containedInSegmentGroupInstance]( Y, SG),
                 DL[D01B:contains](SG, Y),
                 DL[INVOIC_LineItemItemNumber](Y),
                 DL[D01B:hasValue](Y, "xxxxxxxxxxxx"),
                 DL[D01B:hasValue](X, Q),
                 DL[D01B:hasValue](Y, N).
```

P_2 *get all ordered quantities*

```
q(X, Y) :- DL[D01B:ORDERS_LineItemQuantity_orderedquantity](X), DL[D01B:hasValue](X,Y)
q(X, Y) :- DL[D96A:ORDERS_LineItemQuantity_orderedquantity](X), DL[D96A:hasValue](X,Y)
```

P_3 *get requested delivery date/time in order messages and related actual delivery date/time in invoice messages (order number (in ORDERS messages) must equal to the reference order number (in INVOIC messages))*

```
q(X, A1, W, A2) :- order(X, A1, ON), invoice(W, A2, ON).
order(X, A1, ON) :- DL[ORDERS_DateTime_deliverydatetimerequested](X) ,
                    DL[D96A:hasValue](X, A1),
                    DL[D96A:containedInSegmentInstance](X, S),
                    DL[D96A:containedInMessage](S, MSG),
                    DL[D96A:contains](MSG, Y),
                    DL[ORDERS_DocumentNumber](Y),
                    DL[D96A:hasValue](Y, ON).
invoice(W, A2, ON) :- DL[INVOIC_ReferenceIdentifier_ordernumberpurchase] (Z),
                      DL[D01B:hasValue](Z, ON),
                      DL[D96A:containedInSegmentInstance](Z, SI) ,
                      DL[D96A:containedInMessage](SI, MSGI),
                      DL[D96A:contains](MSGI, W),
                      DL[INVOIC_DateTime_deliverydatetimeactual](W),
                      DL[D01B:hasValue](W, A2).
```

P_4 *get orders without related invoices*

```
temp(X) :- DL[D96A:Message](X), DL[D96A:isOfMessageType](X, "ORDERS"),
           DL[D96A:contains](X, OON), DL[ORDERS_DocumentNumber](OON),
           DL[D96A:hasValue](OON, OV), DL[D01B:Message](Y),
           DL[D01B:isOfMessageType](Y, "INVOIC"), DL[D01B:contains](Y, ION),
           DL[INVOIC_ReferenceIdentifier_ordernumberpurchase](ION),
           DL[D01B:hasValue](ION, IV), OV=IV.
q(X) :- not temp(X), DL[D96A:Message](X), DL[D96A:isOfMessageType](X,"ORDERS").
```

P_5 *get invoices without related orders*

```
temp(Y) :- DL[D96A:Message](X), DL[D96A:isOfMessageType](X, "ORDERS"),
           DL[D96A:contains](X, OON), DL[ORDERS_DocumentNumber](OON),
           DL[D96A:hasValue](OON, OV), D01B:Message(Y),
           DL[D01B:isOfMessageType](Y, "INVOIC"), DL[D01B:contains](Y, ION),
           DL[INVOIC_ReferenceIdentifier_ordernumberpurchase](ION),
           DL[D01B:hasValue](ION, IV), OV=IV.
q(Y) :- not temp(Y), DL[D01B:Message](Y), DL[D01B:isOfMessageType](Y, "INVOIC").
```

Table 7.13: dl-programs in the EDI benchmark suite

	P_1	P_2	P_3	P_4	P_5
EDI_5	1.3	1.3	1.3	1.4	1.4
EDI_10	1.8	1.8	1.8	1.8	1.9
EDI_25	4.2	4.0	4.0	4.0	4.1
EDI_50	7.9	7.6	7.7	7.7	7.8
EDI_75	9.4	9.5	9.4	9.7	9.7
EDI_100	11.4	11.3	11.0	11.5	11.5
EDI_125	11.9	11.6	12.1	12.3	12.3
EDI_150	14.8	15.2	15.0	15.4	15.7
EDI_175	16.4	16.3	16.4	16.6	16.7
EDI_200	18.9	17.8	18.1	20.0	20.1

Table 7.14: Evaluation of EDI benchmark suite using DReW[RL]

(5) Policy Benchmark Suite

$$
\begin{aligned}
Deny^{+}(X) &\leftarrow DL[\lambda; UserRequest](X),\ \textit{not}\ DL[\lambda'; \neg Deny](X)\\
Grant^{+}(X) &\leftarrow DL[\lambda; StaffRequest](X),\ \textit{not}\ DL[\lambda'; BlacklistedStaffRequest](X)\\
Deny^{+}(X) &\leftarrow DL[\lambda; BlacklistedStaffRequest](X).\\
in\_Deny(X) &\leftarrow \textit{not}\ out\_Deny(X)\\
out\_Grant(X) &\leftarrow \textit{not}\ in\_Grant(X)\\
fail &\leftarrow DL[\lambda'; Deny](X), out\_Deny(X),\ \textit{not fail}\\
fail &\leftarrow DL[\lambda; Deny](X), in\_Deny(X),\ \textit{not fail}\\
fail &\leftarrow DL[\lambda; Deny](X), out\_Deny(X),\ \textit{not fail}\\
fail &\leftarrow DL[\lambda'; Grant](X), out\_Grant(X),\ \textit{not fail}\\
fail &\leftarrow DL[\lambda; Grant](X), in\_Grant(X),\ \textit{not fail}\\
fail &\leftarrow DL[\lambda; Grant](X), out\_Grant(X),\ \textit{not fail}
\end{aligned}
$$

where $\lambda' = \{Deny \uplus in\_Deny,\ Grant \uplus in\_Grant\}$, and
$\lambda = \{Deny \uplus Deny^{+},\ Grant \uplus Grant^{+}\}$.

Table 7.15: dl-programs in the policy benchmark suite

In the test, we created ontology instances L_i, $i \in \{1, 5, 10, 25\}$, that have a fixed TBox and increasing ABoxes with $i*1000$ instances of user requests. For experimentation, we have populated the ABox with varying numbers of individuals, with varying properties: 1/3 of users are normal users; 1/3 are non-blacklisted staff, and 1/3 are blacklisted staff.

The query imposed was then whether a set of particular individuals, designated by concepts Q_k, $k \in \{5, 50, 100\}$, are granted access (under answer set semantics); the application of defaults, using Q_k as a typing concepts, is thus restricted to the k queried individuals. As we see in Table 7.16, DReW[EL] scales sublinearly in this experiment, on top of both DLV and clingo.

DLVHEX internally transforms the default logic KB (L, D) to the dl-program $(L, \Pi(D))$ as in Table 7.15. This dl-program is highly recursive through the dl-inputs and is quite challenging for DLVHEX. Our tests show that DLVHEX with DF-front end can only handle up to 5 requests (Δ_1 in Table 7.16 already has 1000 request) and runs almost 3 mins. So we did not put the result of dlvhex into the result table.

KB	Typing	DReW[EL]	
		DLV	clingo
Δ_1	5	1.1	0.8
	50	2.4	1.3
	100	6.0	3.0
Δ_5	5	6.6	4.4
	50	8.3	5.0
	100	12.2	7.4
Δ_{10}	5	13.9	9.4
	50	15.7	10.1
	100	20.5	13.3
Δ_{25}	5	35.8	26.0
	50	40.0	26.4
	100	43.7	32.7

Table 7.16: Evaluation of the Policy Benchmark Suite (Time in second)

7.3 Discussion

In this chapter, we have conducted an extensive evaluation of the reasoning of dl-programs on five novel benchmark suites by DReW and DLVHEX system. The results show that DReW always outperforms DLVHEX. Regarding the two rewritings used in DReW, DReW[RL] is better on most of the instances. The results indicate that inline evaluation strategy indeed improves the performance of evaluating dl-programs.

There are several reasons contributing to the bad performance of DLVHEX in these evaluations. (1) When DLVHEX needs to call RacerPro multiple times, we see that there is a big overhead even the DL reasoning is very simple. (2) When multiple calls are needed, RacerPro is easily running out memory because of the multiple copies of the ontologies. (3) Sometimes when DLVHEX needs to call RacerPro with a large ontology, we see that RacerPro had bad performance. When we have to deal with data properties, such performance problem is more visible. This might be a known issue with the version 1.9.2 of RacerPro. By the time of writing this thesis, RacerPro v2 is still not stable. Hopefully the new version will fix this. Observing these points, DLVHEX could be improved by trying coupling with other DL reasoners or the newer version of RacerPro. However, changing DL reasoners still cannot fix the inefficiency caused the overhead of DL calls.

Compared to DLVHEX, the DReW system improves the efficiency a lot using the inline evaluation. However, we still see that there are big rooms for improvements. For instance, it is in general unacceptable to wait several minutes for results of planning in a real geographic information system. There are several further possible optimizations such as caching Datalog rewritings of ontologies. Furthermore we note that in some applications only a small subset of ABox is relevant to the results, e.g., objects in a certain bounding box in some geo information systems. Such application dependent properties can also be exploited for optimizations.

CHAPTER 8

Summary and Outlook

8.1 Summary

The main motivation of this thesis was to improve the performance of reasoning over hybrid KBs, which allow combining KBs formulated in different logics. As a prominent example, dl-programs are a well-studied KR language of combining ASP and DL reasoning in a flexible yet powerful way. Because of the loose coupling nature of dl-programs, one can build engines for dl-programs on top of legacy reasoners. For instance, the DLVHEX system with dl-plugin, which is a state-of-the-art system for dl-programs, is built on top of the ASP reasoner (DLV or Clingo) and the DL reasoner RacerPro. Although this architecture is very elegant, the performance of this implementation is suboptimal.

Observing that the overhead of calling external reasoners in the classical approach can be the bottleneck of the performance, we proposed a new evaluation strategy called inline evaluation by "compiling" the hybrid knowledge base into a knowledge base of a single formalism. In case of dl-programs, we designed an abstract framework rewriting the dl-programs to ASP, by compiling every components (dl-rules, ontology, and dl-atoms) carefully and put them together into a single ASP program. The reduction is sound and complete for the "Datalog-rewritable" DLs with respect to ground queries under both answer set and well-founded semantics. We showed that many DLs, e.g., $\mathcal{LDL}^+$, OWL 2 RL, $\mathcal{SROEL}(\sqcap, \times)$ (OWL 2 EL), and Horn-$\mathcal{SHIQ}$ are Datalog-rewritable by introducing concrete rewriting algorithms.

We further showed that inline evaluation can be used in hybrid KBs of other formalisms. First, terminological default logic KBs and dl-safe conjunctive queries can be reduced to the dl-programs, which then can be inline evaluated when the DL component is Datalog-rewritable. Next, the general conjunctive queries over Horn-$\mathcal{SHIQ}$ can be reduced to the evaluation of Datalog programs via query rewriting. Then this query rewriting algorithm can be naturally extended to more general weakly-safe KBs. Finally, using the rewriting techniques of conjunctive queries, we can extend the inline evaluation to cq-programs.

To confirm the hypothesis that the inline evaluation is superior to the classical approach of "ASP + external DL reasoner", we implemented most of the algorithms of this thesis in the

DReW system, which is a open source reasoner for dl-programs with additional front ends for dl-safe conjunctive queries and terminological default theory. We did extensive evaluations on several novel benchmark suites and showed that DReW system outperforms DLVHEX in general, especially when DLVHEX needs multiple access to RacerPro or the dl-programs are with large ABoxes.

8.2 Outlook

We outlook possible future works and challenges raised by the inline evaluation.

Optimization of the DReW system. Although DReW shows promising performance in the experiments, there are still opportunities for further enhancement on the implementation.

- We observed that a large amount of time is spent on the DL rewriting and grounding. Since the rewriting of the DL ontology is independent of the dl-rules, we could cache the writing results of the DLs for future use and even the grounding.
- DReW is a Java program but DLV itself is not implemented in Java. Currently, we invoke DLV from Java by DLVWrapper which starts a new process of DLV. Then DLV needs to parse the rewritten $\mathsf{Datalog}^{\neg}$ program and do the actual reasoning. The overhead of creating new process, parsing and inter-process communication can be significant and should be reduced. One possibility is to build tighter integration of ASP reasoner (e.g., Clingo or DLV) via Java Native Interface (JNI) and let them share the same data structure of the $\mathsf{Datalog}^{\neg}$ programs in memory.

Recently the DLV team is developing a "DLV Server" which can possibly be accessed remotely (as a DBMS server) and be able to maintain a sort of session information where e.g., the grounding can be shared by several calls [1]. This would be particularly useful in the setting of the DReW system for reducing the overhead of creating DLV instances and grounding Datalog programs.

Backend Engines Now the inline evaluation framework uses ASP as the backend engine. We could think of other backend engines (such as relational database management system (RDBMS) or $\text{DLV}^{\exists}$) for better performance or for more expressivity.

- To use RDBMS was considered in the MOR system [Sch10] to gain scalability. However, RDBMS is not as expressive as ASP, and MOR is limited to the first-order rewritable DL-Lite ontologies. One possibility for supporting more expressive DLs is to use the approximation techniques [BCR10].
- $\text{DLV}^{\exists}$ is a reasoner for (a large fragment of) $\mathsf{Datalog}^{\exists}$, which is the extension of Datalog allowing existentially quantified variables in rule heads [Leo+12]. $\text{DLV}^{\exists}$ is an attractive backend for dl-programs because $\mathsf{Datalog}^{\exists}$ is very expressive and can already capture a large fragment of DLs naturally without complicated query rewriting.

[1] This message is from the personal email communication to the DLV team

Supporting more dl-programs. Note that our current inline evaluation does not work on all dl-programs. The ontologies are restricted to Datalog-rewritable DLs and the dl-atoms can only use $\uplus$. As we discussed in Section 4.6, we could extend our framework by considering the following:

- To support more DLs, we need to discover more Datalog-rewritable DLs. The most expressive Datalog-rewritable logic we know is Horn-$\mathcal{SROIQ}$ [ORS10]. We note that for expressive Datalog-rewritable DLs, the rewriting step is in general intractable so the optimization will be important. Going beyond Horn logics is even more challenging. Grau *el al.* proved that it is impossible in general to answer queries over non-Horn ontologies using Datalog rewritings even for very simple ontology languages, and even if $\mathsf{P} = \mathsf{NP}$ [Gra+13]. We may need to relax Datalog-rewritability to Datalog$^{\vee}$-rewritability and extend our framework accordingly.
- Our current algorithm only supports dl-atoms with the operator $\uplus$. The operator $\uplus$ could be supported by replacing all $C \uplus p$ to $C \uplus \neg p$. Supporting $\Cap$ is possible by eliminating non-monotonic dl-atoms using the techniques in [Wan+13].

More Front Ends Although dl-programs are a very expressive logic formalism, it is not always suitable as a query language to end users. For example, people often don't know how to use the feature of dl-inputs. We could think of dl-programs as a lower level language and "compile" the languages that are more user-friendly or more domain-oriented to the end users into dl-programs and try to inline evaluate the resulting programs. For instance, terminological default logics can be compiled to dl-programs. We are planning to investigate more formalisms, e.g., high-order reasoning which is implemented in DLVHEX and regular path queries over DLs [CEO09; BOS13].

References

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995 (pages 22, 28).

[Acc+05] Andrea Acciarri, Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati. "QuOnto: Querying Ontologies". In: *AAAI*. 2005, pp. 1670–1671 (page 27).

[AKS09] José Júlio Alferes, Matthias Knorr, and Terrance Swift. "Queries to Hybrid MKNF Knowledge Bases through Oracular Tabling". In: *International Semantic Web Conference*. Vol. 5823. Lecture Notes in Computer Science. Springer, 2009, pp. 1–16 (page 51).

[Ant02] Grigoris Antoniou. "Nonmonotonic Rule Systems on Top of Ontology Layers". In: *International Semantic Web Conference*. Ed. by Ian Horrocks and James A. Hendler. Vol. 2342. Lecture Notes in Computer Science. Springer, 2002, pp. 394–398 (page 47).

[AB07] Grigoris Antoniou and Antonis Bikakis. "DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the Semantic Web". In: *IEEE Trans. Knowl. Data Eng.* 19.2 (2007), pp. 233–245 (page 47).

[Ant+01] Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. "Representation results for defeasible logic". In: *ACM Trans. Comput. Log.* 2.2 (2001), pp. 255–287 (page 47).

[Ant+05] Grigoris Antoniou, Carlos Viegas Damasio, Benjamin Grosof, Ian Horrocks, Michael Kifer, Jan Maluszynski, and Peter F. Patel-Schneider. *Combining Rules and Ontologies. A survey.* Tech. rep. REWERSE Project I3-D3, Mar. 2005 (page 39).

[AB88] Krzysztof R. Apt and Howard A. Blair. "Arithmetic Classification of Perfect Models of Stratified Programs". In: *ICLP/SLP*. 1988, pp. 765–779 (page 31).

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. "Pushing the EL Envelope". In: *Proc. IJCAI*. Morgan-Kaufmann Publishers, 2005, pp. 364–369 (page 23).

[BBL08] Franz Baader, Sebastian Brandt, and Carsten Lutz. "Pushing the EL Envelope Further". In: *Proc. OWLED08DC*. http://ceur-ws.org/Vol-496. 2008 (pages 23, 59).

[Baa+07] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. Second. Cambridge University Press, 2007 (page 1).

[BH95] Franz Baader and Bernhard Hollunder. “Embedding Defaults into Terminological Knowledge Representation Formalisms”. In: *J. Autom. Reasoning* 14.1 (1995), pp. 149–180 (pages 3, 90, 105).

[BS96] Franz Baader and Ulrike Sattler. “Number Restrictions on Complex Roles in DLs: A Preliminary Report”. In: *Proc. KR*. 1996, pp. 328–339 (page 71).

[Bar02] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2002 (page 28).

[BS93] Chitta Baral and V. S. Subrahmanian. “Dualities Between Alternative Semantics for Logic Programming and Nonmonotonic Reasoning”. In: *JAR* 10.3 (1993), pp. 399–420 (page 32).

[Ber94] John Berge. *The EDIFACT standards*. Blackwell Publishers, Inc., 1994 (pages 115, 122).

[BOS13] Meghyn Bienvenu, Magdalena Ortiz, and Mantas Simkus. “Conjunctive Regular Path Queries in Lightweight Description Logics”. In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. Ed. by Francesca Rossi. IJCAI/AAAI, 2013 (page 139).

[BFS11] Piero A. Bonatti, Marco Faella, and Luigi Sauro. “Adding Default Attributes to EL++”. In: *AAAI*. Ed. by Wolfram Burgard and Dan Roth. AAAI Press, 2011 (page 110).

[BCR10] Elena Botoeva, Diego Calvanese, and Mariano Rodriguez-Muro. “Expressive Approximations in DL-Lite Ontologies”. In: *Artificial Intelligence: Methodology, Systems, and Applications, 14th International Conference, AIMSA 2010, Varna, Bulgaria, September 8-10. 2010. Proceedings*. Ed. by Darina Dicheva and Danail Dochev. Vol. 6304. Lecture Notes in Computer Science. Springer, 2010, pp. 21–31 (page 138).

[BG04] Dan Brickley and R.V. Guha, eds. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Rec. 10 Feb 2004, http://www.w3.org/TR/rdf-schema/. W3C, 2004 (page 24).

[Bru+07] Jos de Bruijn, David Pearce, Axel Polleres, and Agustín Valverde. “Quantified Equilibrium Logic and Hybrid Rules”. In: *RR*. Ed. by Massimo Marchiori, Jeff Z. Pan, and Christian de Sainte Marie. Vol. 4524. Springer, 2007, pp. 58–72 (page 52).

[Bry+07] François Bry, Norbert Eisinger, Thomas Eiter, Tim Furche, Georg Gottlob, Clemens Ley, Benedikt Linse, Reinhard Pichler, and Fang Wei. “Foundations of Rule-Based Query Answering”. In: *Reasoning Web*. Ed. by Grigoris Antoniou, Uwe Aßmann, Cristina Baroglio, Stefan Decker, Nicola Henze, Paula-Lavinia Patranjan, and Robert Tolksdorf. Vol. 4636. Springer, 2007, pp. 1–153 (pages 7, 32, 35, 71).

[CGL09] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. “Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints”. In: *ICDT’09*. St. Petersburg, Russia: ACM, 2009, pp. 14–30 (page 76).

[Cal+10] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. “Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications”. In: *LICS*. IEEE Computer Society, 2010, pp. 228–242 (pages 51, 86).

[CEO09] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. “Regular Path Queries in Expressive Description Logics with Nominals”. In: *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*. Ed. by Craig Boutilier. 2009, pp. 714–720 (page 139).

[Cal+07] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. “Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family”. In: *JAR* 39.3 (2007), pp. 385–429 (pages 22, 36, 105).

[CGL98] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. “On the Decidability of Query Containment under Constraints”. In: *PODS*. Ed. by Alberto O. Mendelzon and Jan Paredaens. ACM Press, 1998, pp. 149–158 (page 37).

[Car+04] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. “Jena: implementing the semantic web recommendations”. In: *WWW (Alternate Track Papers & Posters)*. Ed. by Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills. ACM, 2004, pp. 74–83 (pages 25, 111).

[CH89] Pohua Chang and Wen-mei Hwu. “Inline function expansion for compiling C programs”. In: *SIGPLAN Not.* 24.7 (June 1989), pp. 246–257 (page 54).

[Che11] Jao-Hong Cheng. “Inter-organizational relationships and information sharing in supply chains”. In: *International Journal of Information Management* 31.4 (2011), pp. 374–384 (page 112).

[CTS11] Alexandros Chortaras, Despoina Trivela, and Giorgos Stamou. “Optimized query rewriting for OWL 2 QL”. In: *Proceedings of the 23rd International Conference On Automated Deduction*. CADE’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 192–206 (page 106).

[CF05] Daniel Corsten and Jan Felde. “Exploring the performance effects of key-supplier collaboration: An empirical investigation into Swiss buyer-supplier relationships”. In: *International Journal of Physical Distribution & Logistics Management* 35.6 (2005), pp. 445–461 (page 112).

[Dan61] D. Ronald Daniel. “Management information crisis”. In: *Harvard Business Review* 39.5 (1961), pp. 111–121 (page 112).

[Dan+01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. “Complexity and Expressive Power of Logic Programming”. In: *ACM Computing Surveys* 33.3 (2001), pp. 374–425 (page 32).

[DEK09] Minh Dao-Tran, Thomas Eiter, and Thomas Krennwallner. “Realizing Default Logic over Description Logic Knowledge Bases”. In: *ECSQARU*. Ed. by Claudio Sossai and Gaetano Chemello. Vol. 5590. Lecture Notes in Computer Science. Springer, 2009, pp. 602–613 (pages 90, 105).

[de +09] Jos de Bruijn, Philippe Bonnard, Hugues Citeau, Sylvain Dehors, Stijn Heymans, Roman Korf, Jörg Pührer, and Thomas Eiter. *Combinations of Rules and Ontologies: State-of-the-Art Survey of Issues*. Tech. rep. Ontorule D3.1. http://ontorule-project.eu/. Ontorule Project Consortium, June 2009 (pages 2, 39, 47).

[DT08] Marc Denecker and Eugenia Ternovska. “A logic of non-monotone inductive definitions”. In: *ACM Transactions on Computational Logic* 9, Issue 2 (2008). Article 14, 52 pp. (page 86).

[Dra+09a] Wlodzimierz Drabent, Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner, Thomas Lukasiewicz, and Jan Maluszynski. “Hybrid Reasoning with Rules and Ontologies”. In: *REWERSE*. Ed. by François Bry and Jan Maluszynski. Vol. 5500. Lecture Notes in Computer Science. Springer, 2009, pp. 1–49 (pages 27, 41).

[Dra+09b] Włodzimierz Drabent, Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner, Thomas Lukasiewicz, and Jan Małuszyński. “Hybrid Reasoning with Rules and Ontologies”. In: *Semantic Techniques for the Web: The REWERSE perspective*. Ed. by Francois Bry and Jan Małuszyński. LNCS 5500. Springer, 2009. Chap. 1, pp. 1–49 (page 39).

[Eit+11] T. Eiter, G. Ianni, T. Lukasiewicz, and R. Schindlauer. “Well-founded semantics for description logic programs in the Semantic Web”. In: *ACM Trans. Comput. Log.* 12.2 (2011), p. 11 (pages 2, 40, 41, 44).

[EG95] Thomas Eiter and Georg Gottlob. “On the Computational Cost of Disjunctive Logic Programming: Propositional Case”. In: *Ann. Math. Artif. Intell.* 15.3-4 (1995), pp. 289–323 (page 34).

[EGM97] Thomas Eiter, Georg Gottlob, and Heikki Mannila. “Disjunctive Datalog”. In: *ACM Trans. Database Syst.* 22.3 (1997), pp. 364–418 (page 34).

[Eit+08c] Thomas Eiter, Georg Gottlob, Magdalena Ortiz, and Mantas Simkus. “Query Answering in the Description Logic Horn-SHIQ”. In: *JELIA’08*. Springer, 2008, pp. 166–179 (pages 76, 102).

[EIK09] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. “Answer Set Programming: A Primer”. In: *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*. Vol. 5689. Lecture Notes in Computer Science. Springer, 2009, pp. 40–110 (pages 28, 29).

[Eit+08b] Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner, and Roman Schindlauer. “Exploiting conjunctive queries in description logic programs”. In: *Ann. Math. Artif. Intell.* 53.1-4 (2008), pp. 115–152 (pages 3, 39, 45, 46, 106).

[Eit+04b] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, and Roman Schindlauer. “Well-founded Semantics for Description Logic Programs in the Semantic Web”. In: *Proc. RuleML*. 2004, pp. 81–97 (pages 56, 57, 59).

[Eit+08a] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. “Combining answer set programming with description logics for the Semantic Web”. In: *Artificial Intelligence* 172.12-13 (2008), pp. 1495–1539 (pages 2, 39–41, 44, 55, 90, 105, 117).

[Eit+04a] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. “Nonmonotonic Description Logic Programs: Implementation and Experiments”. In: *LPAR*. Ed. by Franz Baader and Andrei Voronkov. Vol. 3452. Springer, 2004, pp. 511–527 (pages 44, 117, 120, 123).

[Eit+05] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans 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-05)*. Ed. by Leslie Pack Kaelbling and Alessandro Saffiotti. Professional Book Center, 2005, pp. 90–96 (page 2).

[Eit+06] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. “Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning”. In: *ESWC*. Vol. 4011. LNCS. Springer, 2006, pp. 273–287 (page 45).

[EOS12] Thomas Eiter, Magdalena Ortiz, and Mantas Simkus. “Conjunctive query answering in the description logic SH using knots”. In: *J. Comput. Syst. Sci.* 78.1 (2012), pp. 47–85 (page 76).

[Eng+11] Robert Engel, Worarat Krathu, Marco Zapletal, Christian Pichler, Wil M.P. van der Aalst, and Hannes Werthner. “Process Mining for Electronic Data Interchange”. In: *E-Commerce and Web Technologies*. Ed. by Christian Huemer and Thomas Setzer. Springer, Berlin/Heidelberg, 2011, pp. 77–88 (page 112).

[Eng+12] Robert Engel, Christian Pichler, Marco Zapletal, Worarat Krathu, and Hannes Werthner. “From Encoded EDIFACT Messages to Business Concepts Using Semantic Annotations”. In: *14th IEEE Int. Conf. on Commerce and Enterprise Computing (CEC’12)*. IEEE, 2012 (page 115).

[Fab13] Wolfgang Faber. “Answer Set Programming”. In: *Reasoning Web*. Ed. by Sebastian Rudolph, Georg Gottlob, Ian Horrocks, and Frank van Harmelen. Vol. 8067. Springer, 2013, pp. 162–193 (page 28).

[Feb+12] Onofrio Febbraro, Nicola Leone, Kristian Reale, and Francesco Ricca. “Extending ASPIDE with User-defined Plugins”. In: *CILC*. Ed. by Francesca A. Lisi. Vol. 857. CEUR Workshop Proceedings. CEUR-WS.org, 2012, pp. 236–240 (page 118).

[FRR11] Onofrio Febbraro, Kristian Reale, and Francesco Ricca. “ASPIDE: Integrated Development Environment for Answer Set Programming”. In: *LPNMR*. Ed. by James P. Delgrande and Wolfgang Faber. Vol. 6645. Lecture Notes in Computer Science. Springer, 2011, pp. 317–330 (page 118).

[Geb+12] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012 (page 35).

[GRS91] A. Van Gelder, K. Ross, and J. S. Schlipf. "The Well-Founded Semantics for General Logic Programs". In: *JACM* 38.3 (1991), pp. 620–650 (pages 2, 32, 33).

[GL88] M. Gelfond and V. Lifschitz. "The Stable Model Semantics for Logic Programming". In: *Proc. ICLP*. The MIT Press, 1988, pp. 1070–1080 (pages 1, 31).

[GL91] M. Gelfond and V. Lifschitz. "Classical Negation in Logic Programs and Disjunctive Databases". In: *New Generation Computing* 9 (1991), pp. 365–385 (pages 2, 29, 31).

[Gli+08] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. "Conjunctive Query Answering for the Description Logic SHIQ". In: *JAIR* 31 (2008), pp. 157–204 (page 76).

[GO13] Birte Glimm and Chimezie Ogbuji, eds. *SPARQL 1.1 Entailment Regimes*. W3C Recommendation 21 March 2013, http://www.w3.org/TR/sparql11-overview/. 2013 (page 24).

[GOP11] G. Gottlob, G. Orsi, and A. Pieris. "Ontological queries: Rewriting and optimization". In: *(ICDE), IEEE 27th International Conference on Data Engineering 2011*. Apr. 2011, pp. 2–13 (page 106).

[Gra+13] Bernardo Cuenca Grau, Boris Motik, Giorgos Stoilos, and Ian Horrocks. "Computing Datalog Rewritings Beyond Horn Ontologies". In: *IJCAI*. Ed. by Francesca Rossi. IJCAI/AAAI, 2013 (pages 85, 139).

[Gro+03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. "Description Logic Programs: Combining Logic Programs with Description Logic". In: *Proc. WWW 2003*. ACM, 2003, pp. 48–57 (pages 23, 71).

[Gro12] W3C OWL Working Group, ed. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. W3C Recommendation 11 December 2012, http://www.w3.org/TR/owl2-overview/. 2012 (page 24).

[Gro13] W3C OWL Working Group, ed. *SPARQL 1.1 Overview*. W3C Recommendation 21 March 2013, http://www.w3.org/TR/sparql11-overview/. 2013 (page 24).

[GPH04] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. "An Evaluation of Knowledge Base Systems for Large OWL Datasets". In: *International Semantic Web Conference*. Ed. by Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen. Vol. 3298. Lecture Notes in Computer Science. Springer, 2004, pp. 274–288 (page 121).

[Haa+12] Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. "The RacerPro knowledge representation and reasoning system". In: *Semantic Web* 3.3 (2012), pp. 267–277 (page 26).

[HW08] M. Haklay and P. Weber. "OpenStreetMap: User-Generated Street Maps". In: *Pervasive Computing, IEEE* 7.4 (2008), pp. 12–18 (page 121).

[Hey06] S. Heymans. “Decidable Open Answer Set Programming”. PhD Thesis. Theoretical Computer Science Lab (TINF), CS Dept, Vrije Universiteit Brussel, Feb. 2006 (page 51).

[HFE09] S. Heymans, C. Feier, and T. Eiter. “A Reasoner for Simple Conceptual Logic Programs”. In: *RR*. Ed. by Axel Polleres and Terrance Swift. Vol. 5837. Lecture Notes in Computer Science. Springer, 2009, pp. 55–70 (page 51).

[Hey+10] S. Heymans, R. Korf, M. Erdmann, J. Pührer, and T. Eiter. “Loosely Coupling F-Logic Rules and Ontologies”. In: *IEEE/WIC/ACM International Conference on Web Intelligence (WI 2010)*. IEEE Computer Society, 2010, pp. 248–255 (pages 2, 39, 47).

[HB11] Matthew Horridge and Sean Bechhofer. “The OWL API: A Java API for OWL ontologies”. In: *Semantic Web* 2.1 (2011), pp. 11–21 (pages 25, 111).

[HP04a] I. Horrocks and P. F. Patel-Schneider. “Reducing OWL entailment to description logic satisfiability”. In: *Journal of Web Semantics* 1.4 (2004), pp. 345–357 (page 20).

[HP04b] I. Horrocks and P.F. Patel-Schneider. “A proposal for an OWL rules language”. In: *WWW*. ACM, 2004, pp. 723–731 (page 49).

[Hor+04] 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*. W3C Member Submission. World Wide Web Consortium, 2004 (page 39).

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. “The Even More Irresistible SROIQ”. In: *KR*. AAAI Press, 2006, pp. 57–67 (page 19).

[HMS04] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. “Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs”. In: *Proc. of KR*. AAAI Press, 2004, pp. 152–162 (pages 26, 85, 105).

[HMS07] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. “Reasoning in Description Logics by a Reduction to Disjunctive Datalog”. In: *J. Autom. Reasoning* 39.3 (2007), pp. 351–384 (pages 76, 85, 106).

[ISO95] ISO. *Information technology – Programming languages – Prolog – Part 1: General core*. ISO 13211-1:1995. International Organization for Standardization, 1995 (page 27).

[ISO00] ISO. *Information technology – Programming languages – Prolog – Part 2: Modules*. ISO 13211-2:2000. International Organization for Standardization, 2000 (page 27).

[KB96] Michael Kantor and James H. Burrows. *Electronic Data Interchange (EDI)*. Tech. rep. National Institute of Standards and Technology, 1996 (pages 112, 122).

[KN92] Robert S. Kaplan and David P. Norton. “The balanced scorecard – measures that drive performance”. In: *Harvard business review* 70.1 (1992), pp. 71–79 (page 115).

[Kaz09b] Y. Kazakov. “Consequence-Driven Reasoning for Horn $\mathcal{SHIQ}$ Ontologies”. In: *IJCAI*. Ed. by Craig Boutilier. 2009, pp. 2040–2045 (page 85).

[Kaz08] Yevgeny Kazakov. “RIQ and SROIQ Are Harder than SHOIQ”. In: *KR*. Ed. by Gerhard Brewka and Jérôme Lang. AAAI Press, 2008, pp. 274–284 (page 22).

[Kaz09a] Yevgeny Kazakov. “Consequence-Driven Reasoning for Horn SHIQ Ontologies”. In: *IJCAI'09*. 2009, pp. 2040–2045 (pages 75, 76).

[KKS11] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. “Concurrent Classification of EL Ontologies”. In: *International Semantic Web Conference (1)*. Ed. by Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist. Vol. 7031. Lecture Notes in Computer Science. Springer, 2011, pp. 305–320 (page 26).

[KB13] Michael Kifer and Harold Boley. *RIF Overview (Second Edition)*. W3C Working Group Note 5 February 2013, http://www.w3.org/TR/rif-overview/. 2013 (pages 24, 34).

[KLW95] Michael Kifer, Georg Lausen, and James Wu. “Logical Foundations of Object-Oriented and Frame-Based Languages”. In: *J. ACM* 42.4 (1995), pp. 741–843 (pages 47, 86).

[KAH08] Matthias Knorr, José Júlio Alferes, and Pascal Hitzler. “A Coherent Well-founded Model for Hybrid MKNF Knowledge Bases”. In: *ECAI*. Vol. 178. Frontiers in Artificial Intelligence and Applications. IOS Press, 2008, pp. 99–103 (page 51).

[Kow88] Robert A. Kowalski. “The Early Years of Logic Programming”. In: *Commun. ACM* 31.1 (1988), pp. 38–43 (page 27).

[Kra+12] Worarat Krathu, Christian Pichler, Robert Engel, Marco Zapletal, and Hannes Werthner. “Semantic Interpretation of UN/EDIFACT Messages for Evaluating Inter-organizational Relationships”. In: *Advances in Information Technology*. Vol. 344. Communications in Computer and Information Science. Springer, 2012, pp. 81–93 (pages 115, 116).

[KL07] Adila Krisnadhi and Carsten Lutz. “Data Complexity in the EL family of DLs”. In: *Description Logics*. Ed. by Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Anni-Yasmin Turhan, and Sergio Tessaris. Vol. 250. CEUR-WS.org, 2007 (page 85).

[Krö10] Markus Krötzsch. “Efficient Inferencing for OWL EL”. In: *JELIA*. Ed. by Tomi Janhunen and Ilkka Niemelä. Vol. 6341. Lecture Notes in Computer Science. Springer, 2010, pp. 234–246 (pages 72, 73, 114).

[Krö11] Markus Krötzsch. “Efficient Rule-Based Inferencing for OWL EL”. In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. Ed. by Toby Walsh. IJCAI/AAAI, 2011, pp. 2668–2673 (pages 23, 72, 73, 85, 109, 114).

[Krö+11] Markus Krötzsch, Frederick Maier, Adila Krisnadhi, and Pascal Hitzler. “A better uncle for OWL: nominal schemas for integrating rules and ontologies”. In: *WWW*. Ed. by Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar. ACM, 2011, pp. 645–654 (page 49).

[KMR10] Markus Krötzsch, Anees Mehdi, and Sebastian Rudolph. “Orel: Database-Driven Reasoning for OWL 2 Profiles”. In: *Description Logics*. Ed. by Volker Haarslev, David Toman, and Grant E. Weddell. Vol. 573. CEUR-WS.org, 2010 (page 75).

[KRH07] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. “Complexity Boundaries for Horn Description Logics”. In: *AAAI'07*. AAAI Press, 2007, pp. 452–457 (page 75).

[KRH08b] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. “Description Logic Rules”. In: *Proc. ECAI*. IOS Press, 2008, pp. 80–84 (page 49).

[KRH08a] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. “ELP: Tractable Rules for OWL 2”. In: *Proc. ISWC 2008*. 2008, pp. 649–664 (pages 39, 49).

[Leo+12] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. “Efficiently Computable Datalog$^{\exists}$ Programs”. In: *KR*. Ed. by Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith. AAAI Press, 2012 (page 138).

[Leo+06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. “The DLV System for Knowledge Representation and Reasoning”. In: *ACM Transactions on Computational Logic (TOCL)* 7.3 (July 2006) (pages 35, 111).

[LR98] Alon Y. Levy and Marie-Christine Rousset. “Combining Horn Rules and Description Logics in CARIN”. In: *Artif. Intell.* 104.1-2 (1998), pp. 165–209 (pages 2, 48, 100).

[Lif91] Vladimir Lifschitz. “Nonmonotonic Databases and Epistemic Queries”. In: *IJCAI*. Ed. by John Mylopoulos and Raymond Reiter. Morgan Kaufmann, 1991, pp. 381–386 (page 51).

[LZ04] Fangzhen Lin and Yuting Zhao. “ASSAT: computing answer sets of a logic program by SAT solvers”. In: *Artif. Intell.* 157.1-2 (2004), pp. 115–137 (page 34).

[LLL09] Yi Liu, Yadong Luo, and Ting Liu. “Governing buyer–supplier relationships through transactional and relational mechanisms: Evidence from China”. In: *Journal of Operations Management* 27.4 (2009), pp. 294–309 (page 112).

[Llo87] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987 (pages 27–30).

[Luk10] Thomas Lukasiewicz. “A Novel Combination of Answer Set Programming with Description Logics for the Semantic Web”. In: *IEEE Trans. Knowl. Data Eng.* 22.11 (2010), pp. 1577–1592 (page 2).

[Lut+12] Carsten Lutz, Inanc Seylan, David Toman, and Frank Wolter. *The Combined Approach to OBDA: Taming Role Hierarchies Using Filters*. Tech. rep. University of Bremen, 2012 (pages 121, 127).

[LTW09] Carsten Lutz, David Toman, and Frank Wolter. “Conjunctive Query Answering in the Description Logic EL Using a Relational Database System”. In: *IJCAI*. Ed. by Craig Boutilier. 2009, pp. 2070–2075 (page 106).

[MM79] David Maier and Albert Mendelzon. “Testing implications of data dependencies”. In: *ACM Transactions on Database Systems* 4 (1979), pp. 455–469 (page 77).

[MM04] Frank Manola and Eric Mille, eds. *RDF Primer*. W3C Recommendation 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. 2004 (page 24).

[MT91] V. Wiktor Marek and Miroslaw Truszczynski. “Autoepistemic Logic”. In: *J. ACM* 38.3 (1991), pp. 588–619 (page 33).

[MH12] David Carral Martínez and Pascal Hitzler. “Extending Description Logic Rules”. In: *ESWC*. Ed. by Elena Simperl, Philipp Cimiano, Axel Polleres, Óscar Corcho, and Valentina Presutti. Vol. 7295. Lecture Notes in Computer Science. Springer, 2012, pp. 345–359 (page 49).

[Men97] Elliott Mendelson. *Introduction to mathematical logic (4. ed.)* Taylor & Francis, 1997 (page 8).

[MS09] Julian Mendez and Boontawee Suntisrivaraporn. “Reintroducing CEL as an OWL 2 EL Reasoner”. In: *Description Logics*. Ed. by Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler. Vol. 477. CEUR-WS.org, 2009 (page 26).

[Mot06] Boris Motik. “Reasoning in Description Logics using Resolution and Deductive Databases”. PhD thesis. University of Karlsruhe, Karlsruhe, Germany, Jan. 2006 (page 47).

[Mot+08] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz, eds. *OWL 2 Web Ontology Profiles*. W3C Rec. 27 Oct. 2009, http://www.w3.org/TR/owl2-profiles/. W3C, 2008 (pages 22, 23, 84).

[Mot+12] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation. http://www.w3.org/TR/owl2-profiles/: World Wide Web Consortium, 2012 (pages 22, 72).

[MPG12] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau, eds. *OWL 2 Web Ontology Language Direct Semantics (Second Edition)*. W3C Recommendation 11 December 2012, http://www.w3.org/TR/owl2-overview/. 2012 (page 24).

[MR07] Boris Motik and Riccardo Rosati. “A Faithful Integration of Description Logics with Logic Programming”. In: *Proc. IJCAI*. 2007, pp. 477–482 (page 51).

[MR10] Boris Motik and Riccardo Rosati. “Reconciling Description Logics and Rules”. In: *Journal of the ACM* 57.5 (2010), pp. 1–62 (page 39).

[MSS05] Boris Motik, Ulrike Sattler, and Rudi Studer. “Query Answering for OWL-DL with Rules”. In: *Journal of Web Semantics* 3.1 (July 2005), pp. 41–60 (pages 3, 46, 48, 49).

[MSH09] Boris Motik, Rob Shearer, and Ian Horrocks. “Hypertableau Reasoning for Description Logics”. In: *Journal of Artificial Intelligence Research* 36 (2009), pp. 165–228 (page 26).

[Nar+13] Barbara Nardi, Kristian Reale, Francesco Ricca, and Giorgio Terracina. “An Integrated Environment for Reasoning over Ontologies via Logic Programming”. In: *RR*. Ed. by Wolfgang Faber and Domenico Lembo. Vol. 7994. Lecture Notes in Computer Science. Springer, 2013, pp. 253–258 (page 118).

[ORS10] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. “Worst-case Optimal Reasoning for the Horn-DL Fragments of OWL 1 and 2”. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference (KR-10)*. AAAI Press, May 2010, pp. 269–279 (pages 75, 76, 106, 139).

[ORS11] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. “Query Answering in the Horn Fragments of the Description Logics SHOIQ and SROIQ”. In: *IJCAI’11*. IJCAI/AAAI, 2011, pp. 1039–1044 (pages 76, 78).

[Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994 (page 11).

[Par10] David Parmenter. *Key performance indicators (KPI): developing, implementing, and using winning KPIs*. John Wiley & Sons, 2010 (page 115).

[Pea96] David Pearce. “A New Logical Characterisation of Stable Models and Answer Sets”. In: *NMELP*. Ed. by Jürgen Dix, Luís Moniz Pereira, and Teodor C. Przymusinski. Vol. 1216. Lecture Notes in Computer Science. Springer, 1996, pp. 57–70 (page 52).

[PHM09] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. “Efficient Query Answering for OWL 2”. In: *International Semantic Web Conference*. Ed. by Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan. Vol. 5823. Springer, 2009, pp. 489–504 (page 27).

[PMH09] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. “A Comparison of Query Rewriting Techniques for DL-Lite”. In: *Description Logics*. Ed. by Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler. Vol. 477. CEUR Workshop Proceedings. CEUR-WS.org, 2009 (pages 106, 118).

[PMH10] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. “Tractable query answering and rewriting under description logic constraints”. In: *J. Applied Logic* 8.2 (2010), pp. 186–209 (page 106).

[Prz91] Teodor C. Przymusinski. “Stable Semantics for Disjunctive Programs”. In: *New Generation Comput.* 9.3/4 (1991), pp. 401–424 (page 32).

[Rau09] Wolfgang Rautenberg. *A Concise Introduction to Mathematical Logic*. 3rd ed. Springer, Dec. 17, 2009 (page 7).

[Rei87] R. Reiter. “Readings in nonmonotonic reasoning”. In: ed. by Matthew L. Ginsberg. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987. Chap. On closed world data bases, pp. 300–310 (page 36).

[Rei80] Raymond Reiter. “A Logic for Default Reasoning”. In: *Artif. Intell.* 13.1-2 (1980), pp. 81–132 (pages 89, 90).

[Ric03] Francesco Ricca. “A Java Wrapper for DLV”. In: *Answer Set Programming*. Ed. by Marina De Vos and Alessandro Provetti. Vol. 78. CEUR Workshop Proceedings. CEUR-WS.org, 2003 (page 111).

[Roc79] John F Rockart. “Chief executives define their own data needs.” In: *Harvard business review* 57.2 (1979), p. 81 (page 112).

[RC12] Mariano Rodriguez-Muro and Diego Calvanese. “Quest, an OWL 2 QL Reasoner for Ontology-based Data Access”. In: *OWLED*. Ed. by Pavel Klinov and Matthew Horridge. Vol. 849. CEUR-WS.org, 2012 (page 27).

[RKZ13] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. “OBDA with Ontop”. In: *ORE*. Ed. by Samantha Bail, Birte Glimm, Rafael S. Gonçalves, Ernesto Jiménez-Ruiz, Yevgeny Kazakov, Nicolas Matentzoglu, and Bijan Parsia. Vol. 1015. CEUR Workshop Proceedings. CEUR-WS.org, 2013, pp. 101–106 (page 27).

[Ros99] Riccardo Rosati. “Towards expressive KR systems integrating datalog and description logics: preliminary report”. In: *Description Logics*. Ed. by Patrick Lambrix, Alexander Borgida, Maurizio Lenzerini, Ralf Möller, and Peter F. Patel-Schneider. Vol. 22. CEUR-WS.org, 1999 (page 50).

[Ros05] Riccardo Rosati. “On the Decidability and Complexity of Integrating Ontologies and Rules”. In: *Journal of Web Semantics* 3.1 (2005), pp. 41–60 (pages 48, 50).

[Ros06] Riccardo Rosati. “DL+log: Tight Integration of Description Logics and Disjunctive Datalog”. In: *Proc. KR*. 2006, pp. 68–78 (pages 39, 48, 50, 106).

[Ros07] Riccardo Rosati. “On Conjunctive Query Answering in EL”. In: *Description Logics*. Ed. by Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Anni-Yasmin Turhan, and Sergio Tessaris. Vol. 250. CEUR-WS.org, 2007 (page 106).

[Ros12] Riccardo Rosati. “Prexto: Query Rewriting under Extensional Constraints in DL-Lite”. In: *ESWC*. Ed. by Elena Simperl, Philipp Cimiano, Axel Polleres, Óscar Corcho, and Valentina Presutti. Vol. 7295. Lecture Notes in Computer Science. Springer, 2012, pp. 360–374 (page 27).

[RA10] Riccardo Rosati and Alessandro Almatelli. “Improving Query Answering over DL-Lite Ontologies”. In: *KR*. AAAI Press, 2010 (pages 27, 106).

[Ros53] J. Barkley Rosser. *Logic for mathematicians*. New York, McGraw-Hill, 1953 (page 7).

[Sch12] Michael Schneider, ed. *OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition)*. W3C Recommendation 11 December 2012, http://www.w3.org/TR/owl2-overview/. 2012 (page 24).

[Sch10] Patrik Schneider. “Evaluation of Description Logic Programs using an RDBMS”. MA thesis. Vienna University of Technology, Dec. 2010 (pages 118, 138).

[SKH11] Frantisek Simancik, Yevgeny Kazakov, and Ian Horrocks. “Consequence-Based Reasoning beyond Horn Ontologies”. In: *IJCAI*. Ed. by Toby Walsh. IJCAI/AAAI, 2011, pp. 1093–1098 (page 85).

[Sim00] Patrik Simons. “Extending and Implementing the Stable Model Semantics”. PhD thesis. Helsinki University of Technology, Finland, 2000 (page 34).

[Sim+12] Elena Simperl, Philipp Cimiano, Axel Polleres, Óscar Corcho, and Valentina Presutti, eds. *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*. Vol. 7295. Lecture Notes in Computer Science. Springer, 2012.

[Sir+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. “Pellet: A practical OWL-DL reasoner”. In: *J. Web Sem.* 5.2 (2007), pp. 51–53 (page 26).

[SS08] Markus Stocker and Michael Smith. “Owlgres: A Scalable OWL Reasoner”. In: *OWLED*. Ed. by Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler. Vol. 432. CEUR-WS.org, 2008 (page 27).

[Swi04] T. Swift. “Deduction in Ontologies via ASP”. In: *Proc. of LPNMR*. 2004, pp. 275–288 (page 85).

[Syr01] Tommi Syrjänen. *Lparse 1.0 User’s Manual*. Tech. rep. 2001 (page 34).

[Tar55] A. Tarski. “A Lattice-Theoretical Fixpoint Theorem and its Applications”. In: *Pacific Journal of Mathematics* 5 (1955), pp. 285–309 (pages 61, 62).

[TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. “Distributed computing in practice: the Condor experience”. In: *Concurrency Computat. Pract. Exper.* 17.2-4 (2005), pp. 323–356 (page 123).

[Tob00] Stephan Tobies. “The Complexity of Reasoning with Cardinality Restrictions and Nominals in Expressive Description Logics”. In: *JAIR* 12 (2000), pp. 199–217 (page 67).

[Var82] Moshe Y. Vardi. “The Complexity of Relational Query Languages (Extended Abstract)”. In: *STOC*. Ed. by Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber. ACM, 1982, pp. 137–146 (page 33).

[VSS12] Tassos Venetis, Giorgos Stoilos, and Giorgos B. Stamou. “Incremental Query Rewriting for OWL 2 QL”. In: *Description Logics*. Ed. by Yevgeny Kazakov, Domenico Lembo, and Frank Wolter. Vol. 846. CEUR Workshop Proceedings. CEUR-WS.org, 2012 (page 27).

[VDB10] Joost Vennekens, Marc Denecker, and Maurice Bruynooghe. “FO(ID) as an extension of DL with rules”. In: *Annals of Mathematics and Artificial Intelligence* 58.1-2 (Feb. 2010), pp. 85–115 (pages 39, 51, 86).

[VL05] Fabien Viger and Matthieu Latapy. “Efficient and Simple Generation of Random Simple Connected Graphs with Prescribed Degree Sequence”. In: *COCOON*. Ed. by Lusheng Wang. Vol. 3595. Springer, 2005, pp. 440–449 (page 123).

[Wan+04] Kewen Wang, David Billington, Jeff Blee, and Grigoris Antoniou. “Combining Description Logic and Defeasible Logic for the Semantic Web”. In: *RuleML*. Ed. by Grigoris Antoniou and Harold Boley. Vol. 3323. Lecture Notes in Computer Science. Springer, 2004, pp. 170–181 (page 2).

[Wan+13] Yisong Wang, Thomas Eiter, Jia-Huai You, Li-Yan Yuan, and Yi-Dong Shen. “Eliminating Nonmonotonic DL-Atoms in Description Logic Programs”. In: *RR*. Ed. by Wolfgang Faber and Domenico Lembo. Vol. 7994. Springer, 2013, pp. 168–182 (pages 86, 139).

[Wan+10] Yisong Wang, Jia-Huai You, Li-Yan Yuan, and Yi-Dong Shen. “Loop formulas for description logic programs”. In: *TPLP* 10.4-6 (2010), pp. 531–545 (page 2).

[Wan+12] Yisong Wang, Jia-Huai You, Li-Yan Yuan, Yi-Dong Shen, and Mingyi Zhang. “The loop formula based semantics of description logic programs”. In: *Theor. Comput. Sci.* 415 (2012), pp. 60–85 (page 41).

[War77] David H D Warren. “Prolog: The language and its implementation compared with Lisp”. In: *ACM SIGPLAN Notices*. 1977, pp. 109–115 (page 27).

Relevant Publications

[Cha+13] Günther Charwat, Giovambattista Ianni, Thomas Krennwallner, Martin Kronegger, Andreas Pfandler, Christoph Redl, Martin Schwengerer, Lara Spendier, Johannes Peter Wallner, and Guohui Xiao. "VCWC: A Versioning Competition Workflow Compiler". In: *LPNMR*. 2013 (page 123).

[Eit+12a] Thomas Eiter, Thomas Krennwallner, Patrik Schneider, and Guohui Xiao. "Uniform Evaluation of Nonmonotonic DL-Programs". In: *FoIKS'12*. Springer, Mar. 2012, pp. 1–22 (page 4).

[Eit+12b] Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. "Query Rewriting for Horn-SHIQ Plus Rules". In: *AAAI*. Ed. by Jörg Hoffmann and Bart Selman. AAAI Press, 2012 (pages 4, 75, 107).

[Eit+12c] Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. "Towards Practical Query Answering for Horn-SHIQ". In: *Proceedings of the 2012 International Workshop on Description Logics, DL-2012, Rome, Italy, June 7-10, 2012*. Ed. by Yevgeny Kazakov, Domenico Lembo, and Frank Wolter. Vol. 846. CEUR Workshop Proceedings. CEUR-WS.org, 2012 (pages 4, 107).

[HEX10] Stijn Heymans, Thomas Eiter, and Guohui Xiao. "Tractable Reasoning with DL-Programs over Datalog-rewritable Description Logics". In: *ECAI*. Vol. 215. Frontiers in Artificial Intelligence and Applications. IOS Press, 2010, pp. 35–40 (page 4).

[XE11] Guohui Xiao and Thomas Eiter. "Inline Evaluation of Hybrid Knowledge Bases – PhD Description". In: *Proc. 5th International Conference on Web Reasoning and Rule Systems (RR 2011)*. Ed. by Sebastian Rudolph and Claudio Gutierrez. Vol. 6902. Lecture Notes in Computer Science. Springer, 2011, pp. 300–305 (page 4).

[XEH12] Guohui Xiao, Thomas Eiter, and Stijn Heymans. "The DReW System for Nonmonotonic DL-Programs". In: *Proceedings of Joint Conference of the Sixth Chinese Semantic Web Symposium and the First Chinese Web Science Conference (SWWS 2012)*. Shenzhen City, China, Nov. 2012 (page 4).

[XHE10] Guohui Xiao, Stijn Heymans, and Thomas Eiter. "DReW: a Reasoner for Datalog-rewritable Description Logics and DL-Programs". In: *Informal Proc. 1st Int'l Workshop on Business Models, Business Rules and Ontologies (BuRO 2010), Sept. 21, 2010, Bressanone/Italy*. http://ontorule-project.eu/attachments/075_buro2010-proceedings.pdf. 2010 (pages 4, 109).