PAGOdA: Pay-as-you-go Ontology Query Answering Using a Datalog Reasoner

Yujiao Zhou
New College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity 2015

Abstract

Answering conjunctive queries over ontology-enriched datasets is a core reasoning task for many applications of semantic technologies. Conjunctive query answering is, however, computationally very expensive, which has led to the development of query answering procedures that sacrifice either the expressive power of ontology languages, or the completeness of query answers in order to improve scalability. This thesis describes a hybrid approach to query answering over OWL 2 ontologies that combines a datalog reasoner with a fully-fledged OWL 2 reasoner in order to provide scalable "pay-as-you-go" performance. The key feature of this hybrid approach is that it delegates the bulk of the computation to the datalog reasoner and resorts to expensive OWL 2 reasoning only as necessary to fully answer the query. Although the main goal of this thesis is to efficiently answer queries over OWL 2 ontologies, the technical results are more general and the approach is applicable to first-order knowledge representation languages that can be captured by rules allowing for existential quantification and disjunction in the head; the only assumption is the availability of a datalog reasoner and a fully-fledged reasoner for the language of interest, both of which are used as "black boxes". All techniques proposed in this thesis are implemented in the PAGOdA system, which combines the datalog reasoner RDFox and the OWL 2 reasoner HermiT. An extensive evaluation shows that PAGOdA succeeds in providing scalable pay-as-you-go query answering for a wide range of OWL 2 ontologies, datasets and queries.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisors Ian Horrocks and Bernardo Cuenca Grau for their insightful suggestions and comments for research and career, as well as their patience to teach me how to write. I feel so lucky to have both of them as my supervisors. This thesis would not have been possible without their help.

Besides, I am grateful to all the members of the KRR group who contributed to this thesis in different ways. In particular, Yavor Nenov who offered generous help in the integration of PAGOdA with RDFox and helped me a lot in writing, Robert Piro who patiently assisted me on the access to our group repositaries and servers, as well as my officemates Ana Armas Romero and Giorgio Stefanoni who had all kinds of helpful and interesting discussion with me in the office.

Moreover, I would like to thank the assessors of my transfer and confirmation report, Dan Olteanu and Phil Blunsom, for their helpful suggestions and feedback of my work.

Furthermore, my gratitude also goes to Pacific Alliance Scholarship that provides financial support for my DPhil study, as well as New College that takes great care of different aspects of my life.

Finally, this thesis is dedicated to my families and Bo Tang for their continuous love, encouragement in the past years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Linked data is one of the most popular and useful semantic web technologies: it allows for the publishing and deployment of RDF data (Manola and Miller, 2004) on the web, and it is being adopted by an increasing number of data providers and applications. Linked data allows entities from different resources to be connected to each other arbitrarily via HTTP (Fielding et al., 1999), so that interconnections and context between real world entities can be preserved in the data and exploited for better human and machine understanding.

Due to the large volumes of linked data, efficient data access has become a core problem in the development of applications. In recent years, significant progress has been made in the design and development of RDF data management standards and systems: W3C has provided the standard "SPARQL" query language (Prud'hommeaux and Seaborne, 2008) for RDF based on the class of conjunctive queries (Abiteboul et al., 1995), and state-of-the-art systems such as Hexastore (Weiss et al., 2008) and RDF-3X (Neumann and Weikum, 2010) have implemented highly optimised data structures and query answering algorithms. Throughout this thesis, RDF data and SPARQL queries are considered from a logic point of view.[1]

RDFS (Brickley and Guha, 2014) provides additional vocabulary with predefined meaning that supports conceptual modelling for RDF data, enabling the specification of, e.g., subclass relations between groups of entities, and the domain and range of relations. Early systems, such as Jena (McBride, 2001) and Sesame (Broekstra et al., 2002), were designed to store and manipulate data w.r.t. such RDFS schemas.

OWL, a successor of RDFS, was standardised by W3C in 2004 and revised (as OWL 2) in 2009 providing a much richer conceptual schema language (Baader et al., 2003; Horrocks et al., 2003; Cuenca Grau et al., 2008; Motik et al., 2009b). OWL

[1]We treat RDF as a data model for triples denoting facts such as class membership (unary) and role (binary) assertions, but do not focus on RDF specific details such as blank nodes, etc.

ontologies[2] are widely used to enhance the functionality of RDF-based applications, e.g.,: *(i)* to unambiguously specify the meaning of data in the application; *(ii)* to provide additional vocabulary and background knowledge needed for users and machines to interpret the data; and *(iii)* to enrich knowledge bases with information that is not explicitly represented in the data. Datasets enriched with ontologies, called knowledge bases, have been employed in a variety of realistic application domains, including biology, medicine and defence (Osumi-Sutherland et al., 2012; Schulz et al., 2011; Lacy et al., 2005).

Reasoning can be used to support conceptual modelling, e.g., for checking logical consistency and computing implicit subclass relationships (Buchheit et al., 1993; Horrocks et al., 2000; Calvanese et al., 2001). With the arrival of big-data era, however, there are more and more data-centric applications where the focus has shifted from the conceptual schema to data access (Calvanese et al., 1999; Horrocks and Tobies, 2000; Motik and Sattler, 2006). A key reasoning task in such applications is to answer queries (typically SPARQL queries) w.r.t. both a large dataset (typically RDF) and a rich conceptual schema (typically OWL).

This thesis focuses on conjunctive query answering over knowledge bases that combine an OWL 2 ontology with an RDF dataset. Query answering over such knowledge bases is of very high worst-case complexity (Glimm et al., 2008; Eiter et al., 2012), even when measured only w.r.t. the size of the dataset (so-called *data complexity*). Although heavily optimised, existing systems for query answering, such as HermiT (Glimm et al., 2014), Pellet (Sirin et al., 2007) and RacerPro (Haarslev et al., 2012), can process only small to medium size datasets (Sirin et al., 2007; Möller et al., 2013; Wandelt et al., 2010; Kollia and Glimm, 2013) and hence fall far short of that what is demanded in realistic applications. This has led to the development of query answering procedures that sacrifice expressive power of ontology languages or the completeness of query answers in order to improve scalability.

Conjunctive query answering is tractable w.r.t. data complexity for many fragments of OWL 2, three of which were standardised as so-called *profiles* in OWL 2 (Motik et al., 2009a). The OWL 2 QL and OWL 2 EL profiles are based on the DL-Lite (Calvanese et al., 2007) and the $\mathcal{EL}$ (Baader et al., 2005) families of description logics; the OWL 2 RL profile corresponds to a fragment of the rule-based language datalog (Grosof et al., 2003; Dantsin et al., 2001). Conjunctive query answering systems for OWL 2 profiles have been shown to be highly scalable in practice.

[2]Strictly speaking, an OWL ontology can include both conceptual schema and data, but in practice OWL ontologies are often used as conceptual schemas for data stored as RDF.

Existing systems include OWLim (Bishop et al., 2011), Oracle's native inference engine (Wu et al., 2008), RDFox (Motik et al., 2014), Virtuoso (Erling and Mikhailov, 2009), KARMA (Stefanoni et al., 2013), QuOnto (Acciarri et al., 2005), Mastro (Calvanese et al., 2011), Owlgres (Stocker and Smith, 2008), Rapid (Chortaras et al., 2011), Prexto (Rosati, 2012), Ontop (Bagosi et al., 2014), and so on. The more favourable computational properties of these fragments make them natural choices for data-intensive applications, but they also come at the expense of a loss in expressive power. Many ontologies used in applications are not captured by any of the profiles, such as the NCI Thesaurus[3] and FMA[4].

Among all profiles, OWL 2 RL aims at applications that require scalable data reasoning without sacrificing too much expressive power. It is defined based on DLP—the overlap between OWL 2 and datalog (Grosof et al., 2003). RL reasoners and datalog reasoners are typically implemented by means of materialisation, where query answers are computed against a saturated dataset. In contrast to OWL 2, datalog cannot capture *disjunctive knowledge* such as that expressed in the following axiom, which states that every mammal is a herbivore or a meat eater:

$$\mathsf{Mammal}(x) \rightarrow \mathsf{Herbivore}(x) \vee \mathsf{MeatEater}(x) \tag{1.1}$$

nor can it capture *existentially quantified knowledge* such as that expressed in the following axiom, which states that each mammal must eat something:

$$\mathsf{Mammal}(x) \rightarrow \exists y\, \mathsf{eats}(x, y)$$

These restrictions limit the applicability of datalog reasoners in semantic web applications since disjunctive and existentially quantified statements abound in OWL ontologies. For example, NCI contains many disjunctive statements, while ontologies such as SNOMED[5] (Spackman and Campbell, 1998), FMA, and Fly Anatomy[6] contain thousands of existentially quantified statements.

Although the capabilities of reasoners specialised for fragments of OWL 2 are intrinsically limited, they are typically flexible enough to process arbitrary ontologies on a "best efforts" basis, as a reasoner can simply ignore axioms that are outside its target language. In such cases, query answers for conjunctive queries are still guaranteed to be sound (the computed answer set includes only correct answers), but

[3] `http://evs.nci.nih.gov/ftp1/NCI_Thesaurus/`
[4] `http://sig.biostr.washington.edu/projects/fm/`
[5] `http://www.ihtsdo.org/snomed-ct`
[6] `http://obofoundry.org/cgi-bin/detail.cgi?id=fly_anatomy_xp`

may not be complete (the computed answer set may not include all correct answers); thus, the answer set returned by such a reasoner can be thought of as a *lower bound* on the query answers.

This inspires an alternative way to improve scalability—to sacrifice completeness of query answers. Scalable query answering procedures have been developed that exploit scalable reasoning techniques specialised for light-weight languages, at the expense of computing only approximated query answers. For most existing techniques (Pan and Thomas, 2007; Tserendorj et al., 2008), the computed set of answers is sound but incomplete. One way to realise such a procedure is to weaken an input ontology until it falls within one of the OWL 2 profiles, and then to use a scalable procedure for the relevant fragment. As mentioned before, the required weakening can be trivially achieved by throwing away axioms, but more sophisticated techniques may try to reduce or even minimise information loss (Console et al., 2014). Some techniques provide an upper bound by computing a complete but unsound set of answers (Pan et al., 2009; Wandelt et al., 2010). The existing techniques for upper bound query answers are, however, of much higher computational complexity.

1.1 Contributions and Applications

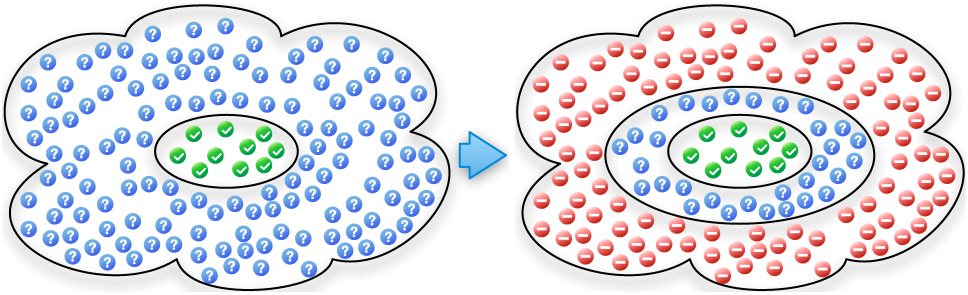
The objective of this research is to develop a practical algorithm to answer conjunctive queries over OWL 2 ontologies. This thesis proposes a sound and complete hybrid approach that combines a scalable datalog reasoner with a fully-fledged OWL 2 reasoner. Given an ontology, a dataset and a query, the hybrid approach uses the datalog reasoner to efficiently compute both lower bound and upper bound query answers. If lower and upper bounds coincide, they obviously provide a sound and complete set of answers. Otherwise, a *relevant subset* of the input ontology and dataset is identified efficiently by relying on the datalog reasoner alone. The key property of this subset is that it is typically much smaller than the input, yet it is still sufficient to test the correctness of answers in the "gap" between lower and upper bounds. Then the gap answers need to be checked w.r.t. the identified subset using the fully-fledged reasoner. Since the number of gap answers can be significant in some cases, more optimisations are exploited to further reduce the number and size of reasoning problems. It is worth mentioning that the approach is also applicable to check satisfiability of the input, which is a prerequisite for query execution.

The key feature of this hybrid approach is its "pay-as-you-go" behaviour: in most cases, the bulk of the computational workload is delegated to the datalog reasoner

while expensive OWL 2 reasoning is used only as necessary to fully answer the query. Furthermore, the extent to which the fully-fledged reasoner is needed will depend on interactions between the ontology, the dataset and the query, so even when the input ontology uses the full power of OWL 2, queries can often be largely or even fully processed using the datalog reasoner. Moreover, this approach has the additional advantage that lower bound answers can be quickly returned, even in cases where completion of the answer requires more time-consuming computations. Although the main goal of this thesis is to efficiently answer conjunctive queries over OWL 2 ontologies and datasets, the technical results presented here are very general and the approach is not restricted to ontology languages based on description logics. More precisely, given a knowledge representation language $\mathcal{L}$ that can be captured by first-order rules, the only assumption for this hybrid approach is the availability of a fully-fledged reasoner for $\mathcal{L}$, which is used as a "black box", and a datalog reasoner.

Improving Lower Bound Answers A datalog reasoner is able to compute a lower bound of query answers by throwing away out-of-profile axioms in the input ontology. Such a lower bound itself is useful in some applications in pressing need for scalable query answering where incompleteness is tolerable. However, the incompleteness of such a lower bound is not evenly distributed—the answers may be complete or nearly complete for some queries and highly incomplete for the others for the same ontology and dataset. In the latter case, it is not unusual for the computed set of answers to be empty. To improve the lower bound, this thesis presents a way to extend an input knowledge base with some deterministic information that can be obtained from disjunctive knowledge. Moreover, due to the monotonicity of conjunctive queries, we further consider different fragments of the input knowledge base and combine the lower bound answers computed for each fragment. Instead of considering these subsets independently, the proposed approach aggregates them to provide a even tighter bound of query answers.

Computing Upper Bound Answers This thesis also proposes several techniques to compute complementary upper bounds by the datalog reasoner. Such upper bounds are useful in practice for (at least) two reasons. First, as illustrated in Figure 1.1b, it allows us to bound the incompleteness of the computed lower bound by partitioning all possible answers into three sets: those that are definitely answers (marked with '✓'), those gap answers that may be answers (marked with '?'), and those that are definitely not answers (marked with '-'). Second, it narrows down candidates that



(a) Lower & trivial bound (b) Lower & upper bound

Figure 1.1: Combination of lower and upper bounds

need to be verified from a potentially huge number of possible answers (as illustrated in Figure 1.1a) to only the gap answers. Please note that the gap can be trivial in some cases, i.e. the lower bound is empty and the upper bound contains all substitutions from answer variables to constants in the knowledge base, however, this is extremely rare. It does not happen in our thorough evaluation and our experiments have shown that in practice the lower and upper bound coincide in most of the cases.

In order to compute a better upper bound of query answers, we introduce new chase techniques that handle both existentially-quantified statements and disjunctive statements to compute finite datasets that preserve query answers w.r.t. the input knowledge base. The upper bounds of query answers computed against each of the datasets can be combined by taking the intersection of them.

The proposed techniques to compute upper bound query answers can be applied to other scenarios rather than query answering, e.g. to generate facet names and values in SemFacet (Arenas et al., 2014).[7] SemFacet is a faceted search system for knowledge bases. In contrast to traditional faceted search systems, SemFacet generates facet names and values dynamically according to the current search result. In this context, a fast response time is critical, and completeness is more important than soundness; this makes dynamic facet generation a perfect application for the upper bound computation.

Extracting Relevant Subsets Although the complexity remains the same to check the validity of gap answers compared with query answering from scratch, the checking times required to call the fully-fledged reasoner can be significantly reduced.

[7]SemFacet is available at `http://www.cs.ox.ac.uk/isg/tools/SemFacet/`.

Sometimes, checking of gap answers may still be too hard in practice. Even satisfiability check for large knowledge bases is not always feasible for a fully-fledged reasoner. On the other hand, it is well-known that query answering in datalog is in polynomial w.r.t. data complexity, which is proved to be efficient in practice. This thesis therefore introduces novel techniques exploiting a datalog reasoner to reduce the size of the knowledge base that should be passed to the fully-fledged reasoner. This is done by a proof tracing technique to identify a *relevant subset* of the input knowledge base that is large enough to check the validity of gap answers and hopefully small enough to be processed by a fully-fledged reasoner. This approach to compute relevant subsets can be reused in other scenarios to reduce the size of reasoning problems. For instance, MORe (Armas Romero et al., 2015) reuses the subset extraction technique for the classification problem in OWL 2 ontologies.

Optimisations before Fully-fledged Reasoning Although, the subset extraction technique allows to reduce the size of the input to a fully-fledged reasoner, it is still very costly to perform verification for each gap answer using a fully-fledged reasoner. This thesis therefore introduces several novel optimisations aiming at further reducing the workload assigned to the fully-fledged reasoner. First, the use of summarisation techniques inspired by the SHER system allows us to quickly identify spurious answers (Dolby et al., 2007, 2009). Afterwards, endomorphism relations and other heuristics are exploited to build a dependency graph between all gap answers. Finally, the dependency graph is used reduce the number of calls to the fully-fledged reasoner. These optimisations are applicable to more general settings. For any conjunctive query answering approach, they can be adopted to pre-filter some spurious query answers and to exploit more information from verified query answers.

The PAGOdA System and Evaluation This pay-as-you-go approach is not only theoretically appealing, but also practically realisable. It has been implemented in PAGOdA using the datalog reasoner RDFox (Motik et al., 2014) and the fully-fledged OWL 2 reasoner HermiT (Glimm et al., 2014). PAGOdA is named as an acronym for "***p**ay-**a**s-you-**g**o **O**WL query **a**nswering system*".[8] This thesis also presents an extensive evaluation using a wide range of realistic and benchmark datasets and queries. In the evaluation, PAGOdA is compared with some of the state-of-the-art query answering engines. The evaluation suggests that PAGOdA is able to deal

[8] A different project with the same name can be found on the website `http://pagoda.lri.fr/` using PAGODA as an acronym for "Practical Algorithm for Ontology-based Data Access".

with complex and expressive ontologies with hundreds of millions of facts, which is well beyond the capabilities of the state-of-the-art: in almost all cases, it is able to completely answer queries without resorting to OWL 2 reasoning, and even when this is not the case, relevant subset extraction and summarisation are effective in reducing the size of the problem to manageable proportions.

In summary, the major contributions of this thesis are listed as follows.

- To the best of my knowledge, it is the first time to present an sound and complete query answering approach that combines two different reasoners, one of which is an efficient datalog reasoner in particular.
- It is novel to take into account existential and disjunctive rules in the computation of query bounds. Even though, the techniques adopted to handle existential and disjunctive rules have been exploited in some other contexts.
- The idea to extract a relevant subset using a datalog encoding appears for the first time.
- It is new to adopt the notation of endomorphism to reduce the number of calls to a fully-fledged reasoner.
- It defines a general class of conjunctive queries that is decidable in OWL 2 and provides a decision procedure for these queries.

1.2 Outline and Structure of this Thesis

This thesis deals with the conjunctive query answering problem over OWL 2 ontologies. The remainder of the thesis is organised as follows.

In Part I, we recapitulate logical foundations that are exploited in this thesis and review existing work on conjunctive query answering.

- This part starts with a recapitulation of definitions and notations in first-order logic in Chapter 2. In particular, we specify a rule-based knowledge representation language and its sub-languages. Since hyperresolution is rather involved in proofs presented later in this thesis, the basic machinery of hyperresolution is introduced at the end of this chapter.

- In Chapter 3, the description logic $\mathcal{SROIQ}$ is introduced, as well as the reasoning tasks over ontologies. Due to the close connection between description logic and OWL ontology language, the web ontology languages OWL 2 is then introduced, as well as the three profiles of it. The choice of using OWL 2 as the target language is mainly due to its rich expressivity and the availability of fully-fledged OWL 2 reasoners. Afterwards, different fragments of OWL 2 are introduced to be exploited for computing approximated query answers.

- Two prominent query languages are discussed in Chapter 4—conjunctive queries and SPARQL queries. For conjunctive queries, we first give its syntax and semantics and then identify classes of conjunctive queries that are decidable for OWL 2 ontologies. Similarly, for SPARQL queries, we first give the syntax and then specify its semantics under OWL 2 Direct Semantics entailment regime.

- Chapter 5, the very last chapter of this part, gives a survey on complexity results of ontological query answering. We also briefly discuss existing query answering techniques for different ontology languages and their corresponding implemented systems.

Part II presents the "pay-as-you-go" approach step by step.

- Chapter 7 presents the techniques to compute and refine lower bound query answers. More precisely, two different treatments are adopted by our approach to improve the basic lower bound computed by discarding non-datalog rules. The two treatments are aimed at extracting some deterministic information from disjunctive and existential rules, respectively.

- Techniques to compute upper bound answers are discussed in Chapter 8 where the basic idea is first presented and followed by two refinements for handling existential and disjunctive rules. The computation of upper bound answers can be done efficiently using a materialisation-based datalog reasoner that supports incremental data addition.

- Chapter 9 explains the intuition and gives an algorithm to compute relevant subsets for queries and gap answers by reusing a datalog reasoner. More optimisations are followed aiming at further reducing the size of extracted subsets or speeding up the extraction process. One of the optimisations is rather technically involved, for which we just give the intuition and a simple example in this chapter. Its definitions and proofs are deferred to Appendix A.

- More optimisations are discussed in Chapter 10 aiming at further reducing the usage of a fully-fledged reasoner. Given a knowledge base, a query, and a set of possible answers to be verified, we adopt two optimisations to reduce the number of possible answers that need to be verified and the number of calls to the fully-fledged reasoner.

- In Chapter 7–9, the approach to compute lower and upper bound query answers is presented in a generic way to deal with arbitrary conjunctive queries. To answer conjunctive queries over OWL 2 ontologies, an OWL 2 reasoner is used in Chapter 10 to verify query answers. Existing OWL 2 reasoners, however, are typically capable to answer only a restricted class of conjunctive queries. Therefore, in Section 10.2, we show how to use an existing OWL 2 reasoner to answer more general class of internalisable queries by reducing internalisable query answering into entailment of data assertions and ontology satisfiability by means of rolling-up.

In Part III, we describe the system PAGOdA implementing the "pay-as-you-go" approach in detail and provide an extensive evaluation.

- Chapter 11 specifies the query language of PAGOdA and provides a systematic description for architecture and work flow of PAGOdA.

- Chapter 12 presents a comprehensive evaluation of PAGOdA. First PAGOdA is compared with some of the state-of-the-art query answering systems on the quality of query answers. Then more experiments are presented to test scalability of PAGOdA and other compared systems. Finally, we conduct experiments to measure the effectiveness of individual techniques implemented in PAGOdA.

Finally, we conclude with discussions on our current approach and present several possibilities for future work.

Some of the techniques and results in this thesis have previously been published and these techniques will not be included in any other thesis. Initially, the idea of computing a complementary upper bound by reusing a datalog reasoner (Zhou et al., 2013b) was published in WWW 2013. Later, a subset extraction technique was developed for Horn ontologies by using backward chaining with tabling (Zhou et al., 2013a), which was published in ISWC 2013. Then the subset extraction approach was extended to arbitrary OWL 2 ontologies and more optimisations were proposed

in the interaction with a fully-fledged reasoner (Zhou et al., 2014) in a work that was published and presented in AAAI 2014. Recently, an extension that combines these three works has been accepted by JAIR.

Part I

Foundations

Chapter 2

Preliminaries

This chapter defines the basic notions needed in this thesis. We start with a recapitulation of first-order logic in Section 2.1. Section 2.2 describes rule languages that will be used throughout this thesis as knowledge representation languages. In Section 2.3, we define the basic machinery for hyperresolution, which will be exploited later as a sound and complete calculus for query answering.

2.1 First-order Logic

In this section, we briefly recapitulate standard syntax and semantics of first-order logic, e.g. see (Ebbinghaus et al., 1996) for a standard textbook. We consider first-order logic without equality, that is, we treat the equality predicate $\approx$ as an ordinary binary predicate without predefined meaning.[1]

Syntax

A *signature* S of a first-order logic language contains the following symbols:

(a) $x, y, z \ldots$ (variables),

(b) $\neg, \wedge, \vee, \rightarrow$ (negation, conjunction, disjunction, and implication),

(c) $\forall, \exists$ (universal and existential quantifiers),

(d) (1) a set N_P^n of n-ary predicates for each $n \geq 0$ with $\bot \in N_P^0$,

(2) a set N_F^n of n-ary function symbols for each $n \geq 1$, and

(3) a set N_C of constants,

where $N_P^0, N_P^1, \ldots, N_F^1, N_F^2, \ldots$, and N_C are pair-wise disjoint.

[1] We choose first-order logic without equality because some existing datalog reasoners do not support native equality reasoning.

We use $\mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots$ to denote vectors of variables. With slight abuse of notation, we sometimes refer $\mathbf{x}$ to the set containing all variables in $\mathbf{x}$ and write $x \in \mathbf{x}$ for each variable x occurring in $\mathbf{x}$.

The set of *S-terms* is as the smallest set such that:

- every variable is an S-term,
- every constant in N_C is an S-term, and
- if $t_1, \ldots, t_n$ are S-terms and $f \in N_F^n$, then $f(t_1, \ldots, t_n)$ is an S-term.

An *S-atomic formula* (or *S-atom*) is in the form of $\bot$ or $P(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are S-terms and $P \in N_P^n$. A *literal* is an atom, called a *positive* literal, or the negation of an atom, called a *negative* literal.

The set of *S-formulas* is defined as the smallest set such that:

- any S-atomic formula is an S-formula,
- if φ is an S-formula, then $\neg\varphi$ is also an S-formula,
- if φ and ψ are S-formulas and $\circ \in \{\wedge, \vee, \rightarrow\}$, then $(\varphi \circ \psi)$ is an S-formula, and
- If φ is an S-formula and x is a variable, then $\forall x\varphi$ and $\exists x\varphi$ are also S-formulas.

We write $\forall\mathbf{x}$(resp. $\exists\mathbf{x}$) as an abbreviation for $\forall x_1 \ldots \forall x_{|\mathbf{x}|}$ (resp. $\exists x_1 \ldots \exists x_{|\mathbf{x}|}$). An occurrence of variable x in a formula φ is *bound* if it belongs to a subformula of the form $\forall x\psi$ or $\exists x\psi$; otherwise, it is free. An *S-sentence* is an S-formula with no free variables. Let Σ be a set of first-order sentences, $\mathsf{Sig}(\Sigma)$ is defined as the smallest signature such that Σ is a set of S-sentences.

A term, an atom or a formula is *ground* if it contains no variables. Moreover, a *fact* is a function-free ground atom.

A formula can contain both free and bound occurrences of the same variable. We can avoid such cases by renaming the bound occurrences of variables. Similarly, we also assume that all the bound variables are different in a formula.

A *substitution* is a mapping from variables to terms which is the identity on all but finitely many variables. The expression $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ (or simply write $\mathbf{x} \mapsto \mathbf{t}$) denotes a substitution σ mapping the variable x_i to t_i for each $1 \leq i \leq n$ and any other variable to itself. The *domain* of σ, written $\mathsf{dom}(\sigma)$, is defined as the set of variables $\{x_1, \ldots, x_n\}$. The *projection* of σ on a vector of variables $\mathbf{y}$ of arity m, written $\sigma|_{\mathbf{y}}$, is defined the substitution $\{y_1 \mapsto \sigma(y_1), \ldots, y_m \mapsto \sigma(y_m)\}$. The result of applying σ on a terms t, written $t\sigma$ (or $t|_{\mathbf{x}\mapsto\mathbf{t}}$), is defined inductively as follows:

- $c\sigma = c$ for a constant $c \in N_C$,
- $x\sigma = \begin{cases} t_i & \text{if } x = x_i \text{ with } 1 \leq i \leq n, \\ x & \text{otherwise,} \end{cases}$
- $(f(t_1, \ldots, t_n))\sigma = f(t_1\sigma, \ldots, t_n\sigma)$ with $f \in N_F^n$.

The result of applying σ on a formula ϕ, written $\phi\sigma$ (or $\phi|_{\mathbf{x}\mapsto\mathbf{t}}$), is defined as follows:

- $\bot\sigma = \bot$,
- $(P(t_1, \ldots, t_n))\sigma = P(t_1\sigma, \ldots, t_n\sigma)$ for $P \in N_P^n$,
- $(\neg\varphi)\sigma = \neg(\varphi\sigma)$,
- $(\varphi \circ \psi)\sigma = \varphi\sigma \circ \psi\sigma$ for $\circ \in \{\wedge, \vee, \rightarrow\}$,
- $(Qx\varphi)\sigma = Qx(\varphi\sigma_x)$ for $Q \in \{\forall, \exists\}$ where $\sigma_x(x) = \begin{cases} \sigma(y) & \text{if } y \neq x, \\ y & \text{if } y = x. \end{cases}$.

The application of a substitution σ can be extended to atoms, sets of terms, sets of atoms, and sets of formulas naturally.

Since we consider first-order logic without equality, the semantics of the equality predicate $\approx$ has to be axiomatised if $\approx$ is include in a signature. For convenience, we write $s \approx t$ for a binary atom $\approx (s, t)$. Let S be a first-order signature including $\approx$. The *equality axiomatisation* of S consists of the following first-order sentences where sentence ($\mathtt{REP_P}$) is instantiated for each n-ary predicate $P \in N_P^n$ and each $1 \leq i \leq n$, and sentence ($\mathtt{REP_F}$) is instantiated for each n-ary function symbol $f \in N_F^n$ and each $1 \leq i \leq n$.

$$\forall x(x \approx x) \qquad (\mathtt{REF})$$

$$\forall x \forall y(x \approx y \rightarrow y \approx x) \qquad (\mathtt{SYM})$$

$$\forall x \forall y \forall z(x \approx y \wedge y \approx z \rightarrow x \approx z) \qquad (\mathtt{TRA})$$

$$\forall \mathbf{x} \forall x_i'(x_i \approx x_i' \wedge P(\mathbf{x}) \rightarrow P(x_1, \ldots, x_i', \ldots, x_n)) \qquad (\mathtt{REP_P})$$

$$\forall \mathbf{x} \forall x_i'(x_i \approx x_i' \rightarrow f(\mathbf{x}) \approx f(x_1, \ldots, x_i', \ldots, x_n)) \qquad (\mathtt{REP_F})$$

Semantics

A *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the domain of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$, which defines the *extension* of constants, function symbols, and predicates such that:

- every constant $c \in N_C$ is mapped to an element $c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

- every function symbol $f \in N_F^n$ is mapped to a function $f^{\mathcal{I}}$ from $(\Delta^{\mathcal{I}})^n$ to $\Delta^{\mathcal{I}}$,
- every predicate $P \in N_P^0$ is mapped to a proposition $P^{\mathcal{I}} \in \{\text{FALSE}, \text{TRUE}\}$ and $\bot^{\mathcal{I}} = \text{FALSE}$, and
- every predicate $P \in N_P^n$ is mapped to $P^{\mathcal{I}}$ a subset of $(\Delta^{\mathcal{I}})^n$.

For any ground term t, we will define recursively the element $t^{\mathcal{I}}$ that is assigned to t by $\mathcal{I}$. If t is a constant in N_C then $t^{\mathcal{I}}$ is part of the interpretation $\mathcal{I}$. For any functional term of the form $f(t_1, \ldots, t_n)$ with $f \in N_F^n$, $t^{\mathcal{I}}$ is defined by $f^{\mathcal{I}}(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}})$. Now we are ready to define $\varphi^{\mathcal{I}}$ for every sentence φ. For each nullary predicate $P \in N_P^0$, $P^{\mathcal{I}}$ is part of the interpretation $\mathcal{I}$. Otherwise, we define

- $(P(t_1, \ldots, t_n))^{\mathcal{I}} = \begin{cases} \text{TRUE} & \text{if } \langle t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}} \rangle \in P^{\mathcal{I}}, \\ \text{FALSE} & \text{if } \langle t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}} \rangle \notin P^{\mathcal{I}}, \end{cases}$
- $(\neg\varphi)^{\mathcal{I}} = \neg(\varphi^{\mathcal{I}})$,
- $(\varphi \circ \psi)^{\mathcal{I}} = \varphi^{\mathcal{I}} \circ \psi^{\mathcal{I}}$ for $\circ \in \{\wedge, \vee, \rightarrow\}$,
- $(\forall x\varphi)^{\mathcal{I}} = \text{TRUE}$ if $\varphi|_{x \mapsto d} = \text{TRUE}$ for all $d \in \Delta^{\mathcal{I}}$, and
- $(\exists x\varphi)^{\mathcal{I}} = \text{TRUE}$ if $\varphi|_{x \mapsto d} = \text{TRUE}$ for some $d \in \Delta^{\mathcal{I}}$.

Let $\mathcal{I}$ be an interpretation, and let φ be a first-order sentence. We say that $\mathcal{I}$ *satisfies (or models)* φ, written $\mathcal{I} \models \varphi$, if $\varphi^{\mathcal{I}} = \text{TRUE}$. Let Ψ be a set of first-order sentences, $\mathcal{I} \models \Psi$ (or $\mathcal{I}$ is a model of Ψ) if $\mathcal{I} \models \psi$ for each $\psi \in \Psi$. The set of sentences Ψ is *satisfiable* if there exists an interpretation satisfying it; otherwise, it is *unsatisfiable*. Ψ *entails* a first-order sentence ϕ (symbolically, $\Psi \models \phi$) if every model of Ψ satisfies ϕ.

Let Ψ_1 and Ψ_2 be sets of first-order sentences. We say that Ψ_2 is a *conservative extension* of Ψ_1 if for every model $\mathcal{I}_1$ of Ψ_1, there exists a model $\mathcal{I}_2$ of Ψ_2 that agrees with $\mathcal{I}_1$ on all the symbols in $\mathsf{Sig}(\Psi_1)$.

If Ψ_2 is a conservative extension of Ψ_1, Ψ_2 preserves all first-order entailments over $\mathsf{Sig}(\Psi_1)$; that is, for each first-order sentence φ over $\mathsf{Sig}(\Psi_1)$, $\Psi_2 \models \varphi$ iff $\Psi_1 \models \varphi$.

Herbrand Interpretations Let S be a first-order signature, the *Herbrand universe* U_S of S is the set of all ground S-terms; the *Herbrand base* B_S is the set of all ground S-atoms. A *Herbrand interpretation* M over S is a subset of B_S with $\bot \notin M$. A Herbrand interpretation M is deemed as the following standard interpretation $\mathcal{I}_M = (\Delta^{\mathcal{I}_M}, .^{\mathcal{I}_M})$.

$$\begin{aligned}
&\Delta^{\mathcal{I}_M} = U_S \\
&c^{\mathcal{I}_M} = c \text{ for each } c \in N_C \\
&f^{\mathcal{I}_M} = \{\langle t_1, \ldots, t_n, f(t_1, \ldots, t_n)\rangle\} \text{ for each } f \in N_F^n \\
&P^{\mathcal{I}_M} = \begin{cases} \text{TRUE} & \text{if } P \in M, \\ \text{FALSE} & \text{if } P \notin M, \end{cases} \quad \text{for each } P \in N_P^0 \\
&P^{\mathcal{I}_M} = \{\langle t_1, \ldots, t_n\rangle \mid P(t_1, \ldots, t_n) \in M\} \text{ for each } P \in N_P^n
\end{aligned}$$

Let Ψ be a set of first-order sentences over S. A Herbrand interpretation M over S is a *Herbrand model* of Ψ if $\mathcal{I}_M \models \Psi$. For an arbitrary set of ground atoms M, it can be seen as a Herbrand interpretation over $\mathsf{Sig}(M)$. When saying that "a set of ground atoms M is a model of Ψ", we actually mean that that the corresponding interpretation $\mathcal{I}_M$ is a model of Ψ.

Finally, we will also exploit the standard notation of homomorphism applicable to sets of atoms, formulas and substitutions.

Let S and T be two set of ground atoms, a *homomorphism* from S to T is a mapping τ from ground terms in S to ground terms in T such that $\varphi \in S$ implies $\varphi\tau \in T$. Let $\mathbf{a}$ and $\mathbf{b}$ be two vectors of ground terms in S of the same arity n, a homomorphism τ from S to S is an *endomorphism* from $\mathbf{a}$ to $\mathbf{b}$ in S if $\tau(a_i) = b_i$ for each $1 \leq i \leq n$. The application of a homomorphism τ can be extended to atoms, formulas and substitutions naturally, e.g. $\alpha\tau$ is the atom $P(t_1\tau, \ldots, t_n\tau)$ for an atom $\alpha = P(t_1, \ldots, t_n)$, and $\sigma\tau$ is the substitution $\{x \mapsto x\sigma\tau \mid x \in \mathsf{dom}(\sigma)\}$ for a substitution σ.

Let Ψ be a set of first-order sentences. A Herbrand model M_1 of Ψ is a *universal model* of Ψ if, for any Herbrand model M_2 of Ψ, M_1 can be homomorphically embedded into M_2.

2.2 Rule-based Knowledge Representation

Rule languages are well-known knowledge representation (KR) formalisms, which have been adopted by deductive databases and logic programming. In this thesis, we restrict ourselves to monotonic rules (i.e. rules without negation as failure).

A *dataset* is a finite set of facts. A *rule* r is a function-free first-order sentence of the form

$$\forall \mathbf{x} \forall \mathbf{y} (\beta_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \beta_n(\mathbf{x}, \mathbf{y}) \rightarrow \bigvee_{i=1}^{m} \exists \mathbf{z}_i \varphi_i(\mathbf{x}, \mathbf{z}_i)) \tag{2.1}$$

where $\mathbf{x}, \mathbf{y}, \mathbf{z}_i$ are pairwise disjoint vectors of variables, $\beta_i(\mathbf{x}, \mathbf{y})$ is an atom different from $\bot$ with free variables in $\mathbf{x} \cup \mathbf{y}$, and either

- $m = 1$ and $\varphi_1(\mathbf{x}, \mathbf{z}_1) = \bot$, or
- $m \geq 1$ and for each $1 \leq j \leq m$ the formula $\varphi_i(\mathbf{x}, \mathbf{z}_j)$ is a conjunction of atoms different from $\bot$ with free variables in $\mathbf{x} \cup \mathbf{z}_j$.

For brevity, the universal quantifiers $\forall \mathbf{x} \forall \mathbf{y}$ are omitted in rules. The conjunction of atoms $\beta_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \beta_n(\mathbf{x}, \mathbf{y})$ is the *body* of r and is denoted by $\mathsf{body}(r)$. The formula $\bigvee_{i=1}^{m} \exists \mathbf{z}_i \varphi_i(\mathbf{x}, \mathbf{z}_i)$ is the *head* of r, denoted $\mathsf{head}(r)$. We assume that rules are *safe*; that is, every variable in $\mathbf{x}$ is mentioned in $\mathsf{body}(r)$. Rules of this form are very general and are able to capture most first-order rule languages for KR such as datalog (Abiteboul et al., 1995), existential rules and datalog$^{\pm}$ (Beeri and Vardi, 1984), as well as datalog$^{\pm,\vee}$ (Alviano et al., 2012b). A rule r is

- *disjunctive datalog* if $\mathsf{head}(r)$ contains no existential quantifiers;
- *existential* or *Horn* if $m = 1$; and
- *datalog* if it is both disjunctive datalog and Horn.

Let $\mathcal{L}$ be a specific rule language. An *$\mathcal{L}$-knowledge base* $\mathcal{K} = \Sigma_{\mathcal{K}} \cup \mathcal{D}_{\mathcal{K}}$ consists of a finite set $\Sigma_{\mathcal{K}}$ of $\mathcal{L}$-rules and a dataset $\mathcal{D}_{\mathcal{K}}$ where each predicate in $\mathcal{D}_{\mathcal{K}}$ is assumed to occur in $\Sigma_{\mathcal{K}}$. We further assume that each knowledge base $\mathcal{K}$ that uses $\approx$ axiomatises its semantics in the usual way; that is, the knowledge base must contain equality axiomatisation of $\mathsf{Sig}(\mathcal{K})$. As the sentence (REF) is not safe, in the equality axiomatisation of a knowledge base, it is replaced by all instances of (REF′) for each n-ary predicate P in $\mathsf{Sig}(\mathcal{K})$ and each $1 \leq i \leq n$.

$$\forall \mathbf{x} (P(\mathbf{x}) \rightarrow x_i \approx x_i) \tag{REF′}$$

Please note the signature of a knowledge base is function-free. So the equality axiomatisation of a knowledge base contains no rules of the form ($\mathtt{REP}_{\mathtt{F}}$).

Normalised Rules In order to simplify the presentation of our technical results, we restrict ourselves to rules a particular normal form, which we specify next.

A rule r is *normalised* if it is of one of the following forms, where $m \geq 1$ and each $\gamma_i(\mathbf{x}, \mathbf{z}_i)$ is a single atom:

$$\beta_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \beta_n(\mathbf{x}, \mathbf{y}) \rightarrow \bot \qquad (\bot\texttt{-Norm})$$

$$\beta_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \beta_n(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}_1 \gamma_1(\mathbf{x}, \mathbf{z}_1) \qquad (\exists\texttt{-Norm})$$

$$\beta_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \beta_n(\mathbf{x}, \mathbf{y}) \rightarrow \gamma_1(\mathbf{x}) \vee \cdots \vee \gamma_m(\mathbf{x}) \qquad (\vee\texttt{-Norm})$$

A knowledge base $\Sigma_{\mathcal{K}} \cup \mathcal{D}_{\mathcal{K}}$ is normalised if all rules in $\Sigma_{\mathcal{K}}$ are normalised.

The restriction to normalised knowledge bases is w.l.o.g. since every set of rules Σ of the form (2.1) can be transformed in polynomial time into a set of normalised rules Σ' that is a conservative extension of Σ as given next.

Proposition 2.1. *Let Σ be a set of rules of the form* (2.1)*. Then there exists a set of normalised rules Σ' that is a conservative extension of Σ and can be computed in polynomial time in the size of Σ.*

Proof. For each rule $r \in \Sigma$ and each $1 \leq i \leq m$, let $\mathbf{x}_i$ be a vector of free variables in the subformulas $\exists \mathbf{z}_i \varphi_i(\mathbf{x}, \mathbf{z}_i)$ of $\mathsf{head}(r)$, then $\mathbf{x}_i \subseteq \mathbf{x}$. Furthermore, let E_{φ_i} be fresh predicates of arity $|\mathbf{x}_i|$ and let C_{φ_i} be fresh predicates of arity $|\mathbf{x}_i| + |\mathbf{z}_i|$ uniquely associated to r and i. Then, Σ' consists of the following rules:[2]

$$\beta_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \beta_n(\mathbf{x}, \mathbf{y}) \rightarrow \bigvee_{i=1}^{m} E_{\varphi_i}(\mathbf{x}_i), \qquad (2.2)$$

$$E_{\varphi_i}(\mathbf{x}_i) \rightarrow \exists \mathbf{z}_i \, C_{\varphi_i}(\mathbf{x}_i, \mathbf{z}_i) \text{ for each } 1 \leq i \leq m, \qquad (2.3)$$

$$C_{\varphi_i}(\mathbf{x}_i, \mathbf{z}_i) \rightarrow \gamma \text{ for each } 1 \leq i \leq m \text{ and each atom } \gamma \text{ in } \varphi_i(\mathbf{x}, \mathbf{z}_i), \qquad (2.4)$$

$$\varphi_i(\mathbf{x}, \mathbf{z}_i) \rightarrow E_{\varphi_i}(\mathbf{x}_i) \text{ for each } 1 \leq i \leq m, \qquad (2.5)$$

$$\varphi_i(\mathbf{x}, \mathbf{z}_i) \rightarrow C_{\varphi_i}(\mathbf{x}_i, \mathbf{z}_i) \text{ for each } 1 \leq i \leq m. \qquad (2.6)$$

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of Σ. We construct an interpretation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ by

[2]Although rules (2.2)–(2.4) are sufficient to form a conservative extension of Σ in normal form, we additionally introduce rules (2.5)–(2.6) in order to facilitate the computation of upper bound query answers (see Section 8.2 and 8.3).

extending $\mathcal{I}$ on those fresh introduced predicates E_{φ_i} and C_{φ_i}. Formally,

$$\begin{aligned}
&\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}};\\
&c^{\mathcal{J}} = c^{\mathcal{I}}, \text{ for each constant } c;\\
&P^{\mathcal{J}} = P^{\mathcal{I}}, \text{ if } P \text{ is in } \mathsf{Sig}(\Sigma);\\
&C^{\mathcal{J}}_{\varphi_i} = \{\mathbf{t} \mid \mathbf{t} \in (\Delta^{\mathcal{I}})^{|\mathbf{x}_i|+|\mathbf{z}_i|} \text{ and } \mathcal{I} \models \varphi_i|_{(\mathbf{x}_i,\mathbf{z}_i)\mapsto\mathbf{t}}\};\\
&E^{\mathcal{J}}_{\varphi_i} = \{\mathbf{t} \mid \mathbf{t} \in (\Delta^{\mathcal{I}})^{|\mathbf{x}_i|} \text{ and } \mathcal{I} \models \exists\mathbf{z}_i\varphi_i|_{\mathbf{x}_i\mapsto\mathbf{t}}\}.
\end{aligned}$$

Let $r \in \Sigma$ be of the form (2.1). We next prove that $\mathcal{J}$ is a model of any rules r' in (2.2)–(2.6) for r.

- If r' is of the form (2.2), let $(\mathbf{a}, \mathbf{b})$ be a tuple such that $|\mathbf{a}| = |\mathbf{x}|$, $|\mathbf{b}| = |\mathbf{y}|$ and $\mathcal{J} \models \{\beta_1(\mathbf{a}, \mathbf{b}), \ldots, \beta_n(\mathbf{a}, \mathbf{b})\}$, and let $\mathbf{a}_i$ be the tuple $\mathbf{x}_i|_{\mathbf{x}\mapsto\mathbf{a}}$. Since $\mathcal{I} \models r$, for some $1 \leq i \leq m$, we have $\mathcal{I} \models \exists\mathbf{z}_i\varphi_i(\mathbf{a}, \mathbf{z}_i)$ and hence $\mathbf{a}_i \in E^{\mathcal{J}}_{\varphi_i}$. So $\mathcal{J} \models r'$.
- If r' is of the form (2.3), let $\mathbf{a}$ be a tuple that $\mathcal{J} \models E_{\varphi_i}(\mathbf{a})$ for some $1 \leq i \leq m$. By the definition of $\mathcal{J}$, we have $\mathcal{I} \models \exists\mathbf{z}_i\varphi_i|_{\mathbf{x}_i\mapsto\mathbf{a}}$. Again by the definition of $\mathcal{J}$, there exists a tuple $\mathbf{e}$ of arity $|\mathbf{z}_i|$ such that $(\mathbf{a}, \mathbf{e}) \in C^{\mathcal{J}}_{\varphi_i}$. So $\mathcal{J} \models r'$.
- If r' is of the form (2.4)–(2.6), the fact $\mathcal{J} \models r'$ is guaranteed by the definition of $C^{\mathcal{J}}_{\varphi_i}$ and $E^{\mathcal{J}}_{\varphi_i}$.

Therefore, $\mathcal{J}$ is a model of Σ' and hence Σ' is a conservative extension of Σ. □

Skolemisation We frequently use Skolemisation to interpret rules in a corresponding Herbrand universe. Skolemisation is formally defined as follows.

Definition 2.2. For each rule r of the form (2.1) and each existentially quantified variable z_{ij}, let f^r_{ij} be a function symbol globally unique for r and z_{ij} of arity $\mathbf{x}$. Furthermore, let θ_{sk} be the substitution such that $\theta_{\mathsf{sk}}(z_{ij}) = f^r_{ij}(\mathbf{x})$ for each $z_{ij} \in \mathbf{z}_i$. The Skolemisation of $\mathsf{sk}(r)$ of r is the following first-order sentence, which by slight abuse of notation we refer to as a *Skolemised rule*:

$$\beta_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \beta_n(\mathbf{x}, \mathbf{y}) \rightarrow \bigvee_{i=1}^{m} \varphi_i(\mathbf{x}, \mathbf{z}_i)\theta_{\mathsf{sk}}$$

Let Σ be a set of rules. The Skolemisation $\mathsf{sk}(\Sigma)$ of Σ is obtained by Skolemising each individual rule in Σ. The definitions of *head* and *body* on rules are extended to skolemised rules naturally. ♢

It is well-known that Skolemisation is an entailment-preserving transformation; that is, for an arbitrary set of rules Σ and any first-order sentence φ in $\mathsf{Sig}(\Sigma)$, $\Sigma \models \varphi$ iff $\mathsf{sk}(\Sigma) \models \varphi$.

Skolem Chase Let $\mathcal{K} = \Sigma_\mathcal{K} \cup \mathcal{D}_\mathcal{K}$ be a Horn knowledge base. There is a universal model of $\mathcal{K}$ that can be characterised by *chase* (Abiteboul et al., 1995; Calì et al., 2013). Here, we use the *Skolem chase* variant (Marnette, 2009; Cuenca Grau et al., 2013).

Definition 2.3. Let $\mathcal{K}$ be a Horn knowledge base. The *Skolem chase sequence* of $\mathcal{K}$ is the sequence of sets of ground atoms $\{\mathcal{B}^i\}_{i\geq 0}$, where $\mathcal{B}^0 = \mathcal{D}_\mathcal{K}$, and $\mathcal{B}^{i+1}$ is inductively defined as follows:

$$\mathcal{B}^i \cup \{\mathsf{head}(\mathsf{sk}(r))\sigma \mid r \in \Sigma_\mathcal{K}, \sigma \text{ a substitution, and } \mathcal{B}^i \models \mathsf{body}(r)\sigma\}.$$

The *Skolem chase* of $\mathcal{K}$, written as $\mathsf{skChase}(\mathcal{K})$, is defined as $\bigcup_{i\geq 0} \mathcal{B}^i$. ♢

The key property of the Skolem chase is that it computes the least Herbrand model of $\mathcal{K}$ if $\mathcal{K}$ is satisfiable. The least Herbrand model $\mathcal{K}$ is a universal model that can be used as a "database" for answering conjunctive queries. Formally, the Skolem chase of $\mathcal{K}$ satisfies that $\mathcal{K}$ is satisfiable iff $\bot \notin \mathsf{skChase}(\mathcal{K})$, and if $\mathcal{K}$ is satisfiable, then $\mathsf{skChase}(\mathcal{K})$ is homomorphically embeddable into any Herbrand model of $\mathcal{K}$ by a homomorphism τ, in such a way that a constants in the domain of τ is mapped to itself. Such a homomorphism τ is called *constant-preserving*. An easy observation of composition between a substitution and a constant-preserving homomorphism will be exploited frequently later in proofs.

Proposition 2.4. *Let τ be a constant-preserving homomorphism between two sets of ground atoms, let β be a atom and let σ be a substitution from free variables in β,*

$$(\beta\sigma)\tau = \beta(\sigma\tau).$$

The relation between Skolem chase and a Herbrand model over Horn knowledge bases can be formulated as follows.

Lemma 2.5. *Let $\mathcal{K} = \Sigma_\mathcal{K} \cup \mathcal{D}_\mathcal{K}$ be a satisfiable Horn knowledge base, and let M be a Herbrand model of $\mathcal{K}$ over an arbitrary signature. There exists a constant-preserving homomorphism τ from $\mathsf{skChase}(\mathcal{K})$ to M*

Proof. Since $\Sigma_\mathcal{K}$ is Horn, any $r \in \Sigma_\mathcal{K}$ is of the form (2.7).

$$\beta_1(\mathbf{x}) \wedge \cdots \wedge \beta_n(\mathbf{x}) \rightarrow \exists y(\gamma_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \gamma_k(\mathbf{x}, \mathbf{y})) \tag{2.7}$$

We first define a function ϵ that maps a rule in $\Sigma_\mathcal{K}$ of the form (2.7) and a tuple of arity $|\mathbf{x}|$ to a tuple of arity $|\mathbf{y}|$ as follows. If $\{\beta_1(\mathbf{a}), \ldots, \beta_n(\mathbf{a})\} \subseteq M$, then $\epsilon(r, \mathbf{a})$ is

defined as a tuple $\mathbf{e}$ such that $\gamma_i(\mathbf{a}, \mathbf{e}) \in M$ for all $1 \leq i \leq k$. Because M satisfies $\mathcal{K}$, ϵ is well-defined. We next define a mapping from terms in $\mathsf{skChase}(\mathcal{K})$ to terms in M as follows. For each term t in $\mathsf{skChase}(\mathcal{K})$,

$$\tau(t) = \begin{cases} t & \text{if } t \text{ is a constant;} \\ (\epsilon(r, \mathbf{t}))_j & \text{if } t \text{ is of the form } f_j^r(\mathbf{t}). \end{cases}$$

Clearly, τ is constant-preserving. So it suffices to prove that τ is a homomorphism from $\mathsf{skChase}(\mathcal{K})$ to M. Let $\mathcal{B}^0, \mathcal{B}^1, \ldots$ be the Skolem Chase sequence of $\mathcal{K}$. We proceed inductively to show that τ is a homomorphism from B^i to M on index i.

Base case: Clearly, $\mathcal{B}^0 = \mathcal{D}_\mathcal{K}$. Since τ is constant-preserving, for each atom $\alpha \in \mathcal{D}_\mathcal{K}$, $\alpha\tau = \alpha \in M$.

Inductive step: Assuming the claim holds for i, we next prove τ is a homomorphism from $\mathcal{B}^{i+1}$ to M. So $\mathcal{B}^i\tau \subseteq M$, and then it suffices to show that $(\mathcal{B}^{i+1} \setminus \mathcal{B}^i)\tau \in M$. Let $\alpha \in \mathcal{B}^{i+1} \setminus \mathcal{B}^i$, there exists a rule r of the form (2.7), a tuple $\mathbf{a}$ such that $\mathcal{B}^i \models \beta_1(\mathbf{a}) \wedge \cdots \wedge \beta_n(\mathbf{a})$ and $\alpha = \gamma_i(\mathbf{a}, \mathbf{y})\theta_{\mathsf{sk}}$ for some $1 \leq i \leq k$ where θ_{sk} maps each y_j in r to $f_j^r(\mathbf{x})$. By the induction hypothesis, $\{\beta_1(\mathbf{a}\tau), \ldots, \beta_n(\mathbf{a}\tau)\} \subseteq M$. Therefore, by the definition of ϵ we know that $\gamma_i(\mathbf{a}\tau, \epsilon(r, \mathbf{a}\tau)) \in M$. For each $1 \leq j \leq |\mathbf{y}|$, $y_j\theta_{\mathsf{sk}}\tau = f_j^r(\mathbf{x})\tau = (\epsilon(r, \mathbf{t}))_j$. Consequently,

$$\alpha\tau = \gamma_i(\mathbf{a}, \mathbf{y})\theta_{\mathsf{sk}}\tau = \gamma_i(\mathbf{a}\tau, \mathbf{y}\theta_{\mathsf{sk}}\tau) = \gamma_i(\mathbf{a}\tau, \epsilon(r, \mathbf{a}\tau)) \in M.$$

Because $\mathsf{skChase}(\mathcal{K}) = \bigcup_{i \geq 0} \mathcal{B}^i$, τ is a homomorphism from $\mathsf{skChase}(\mathcal{K})$ to M. □

Note that $\mathsf{skChase}(\mathcal{K})$ might contain infinitely many atoms. If $\Sigma_\mathcal{K}$ is datalog, however, $\mathsf{skChase}(\mathcal{K})$ is guaranteed to be finite as the Herbrand universe of $\mathcal{K}$ is finite. In this case, we often refer to $\mathsf{skChase}(\mathcal{K})$ as the *materialisation* of $\mathcal{K}$. The materialisation of $\mathcal{K}$ contains precisely all facts logically entailed by $\mathcal{K}$.

2.3 Hyperresolution

Reasoning over knowledge bases can be realised by means of the *hyperresolution calculus* (Robinson and Voronkov, 2001), which we briefly recapitulate in this section.

Let $\{t_1, \ldots, t_n\}$ be a set of terms. A substitution σ is a *unifier* of $\{t_1, \ldots, t_2\}$ if $t_1\sigma = \cdots = t_n\sigma$. A unifier σ of $\{t_1, \ldots, t_n\}$ is a *most general unifier* (MGU) if there exists a substitution τ such that $\sigma\tau = \sigma'$ for any unifier σ' of $\{t_1 \ldots t_n\}$. If a most general unifier exists for a set of terms, it is unique up to renaming (Lloyd, 1984),

and we denote it as $\mathsf{MGU}(t_1, \ldots, t_n)$. And the notion can be extended to atoms and sets of atoms naturally.

A *clause* is a disjunction of literals of the form

$$\neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \gamma_1 \vee \cdots \vee \gamma_m \tag{2.8}$$

where β_i and γ_j are atoms for each $1 \leq i \leq n, 1 \leq j \leq m$. A clause is *positive* if $n = 0$; it is *ground* if it contains no variables. We assume that $\bot$ does not occur in clauses and denote with $\Box$ the empty clause. Furthermore, we treat disjunctions of ground atoms as sets and hence we do not allow for duplicated atoms in a disjunction. The Skolemisation $\mathsf{sk}(r)$ of a normalised rule r is logically equivalent to the clause containing each atom different from $\bot$ in $\mathsf{head}(\mathsf{sk}(r))$ and the negation of each atom in $\mathsf{body}(\mathsf{sk}(r))$, so we sometimes abuse notation and use $\mathsf{sk}(r)$ to refer to a Skolemised rule or its corresponding clause.

Definition 2.6. The inference rule for hyperresolution (Robinson, 1965) is

$$\frac{\chi_i \vee \alpha_1 \;\; \ldots \;\; \chi_n \vee \alpha_n \quad \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \psi}{\chi_1 \vee \cdots \vee \chi_n \vee \psi\sigma}$$

where each χ_i and ψ are positive ground clauses, and $\sigma = \mathsf{MGU}(\alpha_i, \beta_i)$ for each $1 \leq i \leq n$. The positive ground clause $\chi_1 \vee \cdots \vee \chi_m \vee \psi\sigma$ is a *hyperresolvent* of $\chi_1 \vee \alpha_1, \ldots, \chi_n \vee \alpha_n$ and $\neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \psi$. The inference is called a *hyperresolution step*, where the clause $\neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \psi$ is the *main premise*. $\diamondsuit$

Hyperresolution is closely related to ordered resolution with maximal selection. The inference rule of ordered resolution with maximal selection is the same as hyperreolution with additional requirements (Bachmair and Ganzinger, 2001). Therefore ordered resolution inference with maximal selection is a more restrictive and optimised inference system than hyperresolution.

Definition 2.7. Let $\mathcal{K} = \Sigma_\mathcal{K} \cup \mathcal{D}_\mathcal{K}$ be a normalised knowledge base and let χ be a positive ground clause. A *derivation* of χ from $\mathcal{K}$ is a pair $\rho = (T, \lambda)$ where T is a tree, λ is a labeling function that maps each node in T to a ground clause, and for each v in T:

(1) $\lambda(v) = \chi$ if v is the root;

(2) $\lambda(v) \in \mathcal{D}_\mathcal{K}$ if v is a leaf; and

(3) if v has children $w_1, \ldots, w_n$, let $r \in \Sigma_{\mathcal{K}}$ be such that $\mathsf{sk}(r)$ is the main premise in this hyperresolution step to obtain $\lambda(v)$. Then $\lambda(v)$ is a hyperresolvent of $\lambda(w_1), \ldots, \lambda(w_n)$ and $\mathsf{sk}(r)$.

The *support* of ρ, written $\mathsf{support}(\rho)$, is the set of facts in $\mathcal{D}_{\mathcal{K}}$ and rules in $\Sigma_{\mathcal{K}}$ participating in hyperresolution steps in ρ. $\diamondsuit$

We write $\mathcal{K} \vdash \chi$ to denote that there is a derivation of χ from $\mathcal{K}$. Hyperresolution is sound and complete in the following sense: $\mathcal{K}$ is unsatisfiable if and only if $\mathcal{K} \vdash \square$; furthermore, if $\mathcal{K}$ is satisfiable then $\mathcal{K} \models \alpha$ if and only if $\mathcal{K} \vdash \alpha$ for an arbitrary ground atom α.

Chapter 3

Description Logics and Ontologies

In this chapter, we define the syntax and semantics of the Description Logics (DLs) underpinning the W3C standard ontology language OWL 2. Typically, the predicates in DL signatures are restricted to be unary and binary; the former are called concepts and the latter are referred as roles. DLs provide two special concepts $C_\bot$ (the bottom concept) and $C_\top$ (the top concept), which are mapped by every interpretation to the empty set and the interpretation domain, respectively.

3.1 The Logic $\mathcal{SROIQ}$

$\mathcal{SROIQ}$ is the logic underpinning the web ontology language OWL 2. Each axiom in $\mathcal{SROIQ}$ corresponds to a first-order sentence containing only unary and binary predicates. Table 3.1 summarises the naming conversion in DLs, cf. (Baader et al., 2003). For instance, $\mathcal{SROIQ}$ stands for an extension of the DL $\mathcal{ALC}$ with transitivity ($\mathcal{S}$), complex role inclusions ($\mathcal{R}$), nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$) and quantified cardinality constraints ($\mathcal{Q}$).

3.1.1 Syntax and Semantics

Similar to the introduction of first-order logic in Section 2.1, we start with syntax and semantics of $\mathcal{SROIQ}$ (Horrocks et al., 2006).

Syntax

A *signature* of $\mathcal{SROIQ}$ contains the following symbols:

$\mathcal{C}$	$\neg C$ complex concept negation	$\mathcal{I}$	R^- inverse role
$\mathcal{S}$	$\mathcal{ALC}$ with transitivity	$\mathcal{H}$	$R \sqsubseteq S$ role hierarchy
$\mathcal{U}$	$C \sqcup D$ concept disjunction	$\mathcal{O}$	$\{o\}$ nominal
$\mathcal{N}$	$\geq nR, \leq nR$ cardinality constraint		
$\mathcal{Q}$	$\geq nR.C, \leq nR.C$ quantified cardinality constraint		
$\mathcal{R}$	$R_1 \circ \cdots \circ R_n \sqsubseteq R$ complex role inclusion		

Table 3.1: Symbols for DL constructors

(a) $\neg, \sqcap, \sqcup, \sqsubseteq$ (negation, conjunction, disjunction, and implication),

(b) $\forall, \exists,$ (universal and existential quantifiers,
$\geq, \leq$ min and max cardinality),

(c) $^-, \circ$ (inverse, composition),

(d) $\mathsf{Ref}, \mathsf{Irr}, \mathsf{Sym}, \mathsf{Asy},$ (reflexive, irreflexivity, symmetry, asymmetry,
$\mathsf{Tra}, \mathsf{Dis}, \mathsf{Func}$ transitivity, disjointness, functionality),

(e) Self (self constructor),

(f) (1) a set N_P^1 of unary predicate including $C_\top, C_\bot$,
(2) a set N_P^2 of binary predicates,
(3) a set N_C of constants,
where N_C, N_P^1, and N_P^2 are pair-wise disjoint.

In DL terminologies, predicates in N_P^1 other than $C_\bot$ are called *atomic concepts*, predicates in N_P^2 are called *atomic roles*. We treat $C_\bot$ as the negation of $C_\top$.

Roles The set of *roles* is $\{R, R^- \mid R \in N_P^2\}$ where R^- is called the *inverse role* of R. For convenience, we denote by $\mathsf{inv}(R)$ the inverse of a role

$$\mathsf{inv}(R) = \begin{cases} R^- & \text{if } R \text{ is an atomic role,} \\ S & \text{if } R = S^- \text{ with an atomic role } S, \end{cases}$$

and $\mathsf{ar}(R, t_1, t_2)$ the first-order atom for a role R and terms t_1, t_2.

$$\mathsf{ar}(R, t_1, t_2) = \begin{cases} R(t_1, t_2) & \text{if } R \text{ is an atomic role,} \\ S(t_2, t_1) & \text{if } R = S^- \text{ with an atomic role } S. \end{cases}$$

Concepts $\mathcal{SROIQ}$-concepts are formed according to the following gammar:

$$\begin{aligned} C, D \longrightarrow\ & A \mid C_\bot \mid \{a\} \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \\ & \forall R.C \mid \exists R.C \mid \exists S.\mathsf{Self} \mid \geq n\, S.C \mid \leq n\, S.C \end{aligned}$$

where A denotes an atomic concept, a an individual, C, D concepts, S, R roles and n a positive integer.

Axioms A *role axiom* is in one of the forms $\mathsf{Ref}(R)$, $\mathsf{Irr}(R)$, $\mathsf{Sym}(R)$, $\mathsf{Asy}(R)$, $\mathsf{Tra}(R)$ and $\mathsf{Dis}(R, S)$ for roles R and S. A *role inclusion* is of the form $R_1 \circ \cdots \circ R_n \sqsubseteq R$ where $\circ$ denotes composition of roles. A role inclusion is *complex* if $n > 1$. A *general concept inclusion (GCI)* is of the form $C \sqsubseteq D$, where C, D are $\mathcal{SROIQ}$-concepts. An *ontology* consists of a finite set of role axioms, role inclusions and GCIs.

Data Assertions A *data assertion* is in one of the following forms: $C(a)$, $R(a, b)$, or $\neg R(a, b)$ for constants a and b, a concept C, and a role R.[1] Given a data assertion of the form $C(a)$, we say that constant a is an *instance* of concept C. If a set consists of data assertions of the form $A(a)$ for an atomic concept A or $P(a, b)$ for an atomic role P, the set is a dataset. Depending on context, we sometimes refer to an ontology as a finite set of axioms and data assertions.

Syntax Restrictions In order to ensure decidability of reasoning tasks, $\mathcal{SROIQ}$ specialises a number of additional restrictions. First, role hierarchies are restricted to the regular ones, which involve a regular order. A partial ordering $\prec$ on roles is called a *regular order* if $\prec$ satisfies, additionally, $S \prec R \Leftrightarrow \mathsf{inv}(S) \prec R$, for any role R and an atomic role S. A role inclusion $S_1 \circ \cdots \circ S_n \sqsubseteq R$ is $\prec$-regular if R is a role name and *(i)* $n = 1, S_1 = \mathsf{inv}(R)$; or *(ii)* $n = 2, S_1 = S_2 = R$; or *(iii)* $S_1 = R$ and $S_i \prec R$ for each $1 < i \leq n$; or *(iv)* $S_n = R$ and $S_i \prec R$ for each $1 \leq i < n$; or *(v)* $S_i \prec R$ for each $1 \leq i \leq n$. Finally, a role hierarchy is regular if there exists a regular order $\prec$ such that all role inclusions in it are $\prec$-regular. This syntax restriction is a sufficient but not necessary condition for decidability, and a more relaxed condition is described by Kazakov (2010).

Second, the usage of transitive roles are restricted. A role is called simple if it has no transitive subroles, which is formally defined as follows. The set of roles that are *simple* in an ontology $\mathcal{O}$ inductively define as follows: *(i)* an atomic role is simple if it doesn't appear in the right-hand-side of any role inclusion in $\mathcal{O}$; *(ii)* an inverse role R^- is simple if R is; and *(iii)* R is simple if, it only appears in role inclusions of the form $S \sqsubseteq R$ with S a simple role. In $\mathcal{SROIQ}$, roles occurring in role axioms of the

[1]Please note negative concept assertions are implicitly supported as C is complex and it can be the negation of another concept.

forms $\mathsf{Irr}(S)$, $\mathsf{Asy}(S)$ and $\mathsf{Dis}(S_1, S_2)$ and concepts of the forms $\exists S.\mathsf{Self}$, $\geq n\, S.C$ and $\leq n\, S.C$ are restricted to simple ones.

$\mathcal{SROIQ}$ with the above syntax restrictions underpins the web ontology language OWL 2. The logic foundation of its predecessor OWL DL is $\mathcal{SHOIN}$, which is obtained by disallowing complex role inclusions, and restricting cardinality restrictions to be of the form $\geq n\, R.C_\top$ and $\leq n\, R.C_\top$.

Semantics

A first-order interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ over a DL signature defines the extension of atomic concepts, atomic roles and constants, where the extension of $C_\top$ and $C_\bot$ is restricted such that $C_\top^\mathcal{I} = \Delta^\mathcal{I}$ and $C_\bot^\mathcal{I} = \emptyset$.

Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be an interpretation over a $\mathcal{SROIQ}$ signature. For an atomic role R, the extension of R is already defined in $\mathcal{I}$ and the extension of the inverse role R^- is defined as

$$(R^-)^\mathcal{I} = \{\langle b, a\rangle \mid \langle a, b\rangle \in R^\mathcal{I}\}.$$

For any roles R and S, the extension of the composition $R \circ S$ is defined as

$$(R \circ S)^\mathcal{I} = \{\langle a, c\rangle \mid \langle a, b\rangle \in R^\mathcal{I}, \langle b, c\rangle \in S^\mathcal{I}\}.$$

For concepts C, D, roles R, S, and a positive integer n, the extension of atomic concepts are already defined in $\mathcal{I}$, otherwise, the extension of (complex) concepts is defined inductively by the following equations where $\sharp(M)$ denotes the cardinality of the set M.

$$\begin{aligned}
&(\{a\})^\mathcal{I} = \{a^\mathcal{I}\}, \qquad (\neg C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I},\\
&(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}, \qquad (C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I},\\
&(\exists S.\mathsf{Self})^\mathcal{I} = \{a \mid \langle a, a\rangle \in S^\mathcal{I}\},\\
&(\exists R.C)^\mathcal{I} = \{a \mid \exists b\, \langle a, b\rangle \in R^\mathcal{I} \text{ and } b \in C^\mathcal{I}\},\\
&(\forall R.C)^\mathcal{I} = \{a \mid \forall b\, \langle a, b\rangle \in R^\mathcal{I} \text{ implies } b \in C^\mathcal{I}\},\\
&(\geq n\, S.C)^\mathcal{I} = \{a \mid \sharp\{b \mid \langle a, b\rangle \in S^\mathcal{I} \text{ and } b \in C^\mathcal{I}\} \geq n\},\\
&(\leq n\, S.C)^\mathcal{I} = \{a \mid \sharp\{b \mid \langle a, b\rangle \in S^\mathcal{I} \text{ and } b \in C^\mathcal{I}\} \leq n\}.
\end{aligned}$$

For an interpretation $\mathcal{I}$ that satisfies a role axiom, a role inclusion, a GCI or a

data assertion, we have

$$
\begin{aligned}
&\mathcal{I} \models \mathsf{Ref}(R) && \text{if } \langle a, a\rangle \in R^{\mathcal{I}}, \text{ for each } a \in \Delta^{\mathcal{I}};\\
&\mathcal{I} \models \mathsf{Irr}(S) && \text{if } \langle a, a\rangle \notin S^{\mathcal{I}}, \text{ for each } a \in \Delta^{\mathcal{I}};\\
&\mathcal{I} \models \mathsf{Sym}(R) && \text{if } \langle a, b\rangle \in R^{\mathcal{I}} \text{ implies } \langle b, a\rangle \in R^{\mathcal{I}};\\
&\mathcal{I} \models \mathsf{Asy}(S) && \text{if } \langle a, b\rangle \in S^{\mathcal{I}} \text{ implies } \langle b, a\rangle \notin S^{\mathcal{I}};\\
&\mathcal{I} \models \mathsf{Tra}(R) && \text{if } \langle a, b\rangle \in R^{\mathcal{I}} \text{ and } \langle b, c\rangle \in R^{\mathcal{I}} \text{ implies } \langle a, c\rangle \in R^{\mathcal{I}};\\
&\mathcal{I} \models \mathsf{Dis}(S_1, S_2) && \text{if } S_1^{\mathcal{I}} \cap S_2^{\mathcal{I}} = \emptyset;\\
&\mathcal{I} \models R_1 \sqsubseteq R_2 && \text{if } R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}};
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{I} \models C \sqsubseteq D && \text{if } C^{\mathcal{I}} \subseteq D^{\mathcal{I}}; && \mathcal{I} \models R(a, b) && \text{if } \langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}};\\
&\mathcal{I} \models C(a) && \text{if } a^{\mathcal{I}} \in C^{\mathcal{I}}; && \mathcal{I} \models \neg R(a, b) && \text{if } \langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \notin R^{\mathcal{I}}.
\end{aligned}
$$

Let $\mathcal{O}$ be an arbitrary ontology and let $\mathcal{D}$ be a finite set of data assertions. An interpretation $\mathcal{I}$ satisfies (is a model of) $\mathcal{O} \cup \mathcal{D}$, written as $\mathcal{I} \models \mathcal{O} \cup \mathcal{D}$, if $\mathcal{I}$ satisfies each role axiom, each role inclusion, each GCI and each data assertion in $\mathcal{O} \cup \mathcal{D}$. $\mathcal{O} \cup \mathcal{D}$ is *satisfiable* if there exists such a model of it. Let α be either a role axiom, a role inclusion, a GCI or an data assertion, then $\mathcal{O} \cup \mathcal{D} \models \alpha$ if $\mathcal{I} \models \mathcal{O} \cup \mathcal{D}$ implies $\mathcal{I} \models \alpha$ for any interpretation $\mathcal{I}$.

In general, we restrict ourselves to data assertions of the form $A(a)$ and $P(a, b)$ with an atomic concept A and an atomic role P. This is w.l.o.g. since

$$
\begin{gathered}
\mathcal{O} \cup \{A \sqsubseteq C,\ A(a)\} \text{ is a conservative extension of } \mathcal{O} \cup \{C(a)\},\\
\mathcal{O} \cup \{P(b, a)\} \text{ is equivalent to } \mathcal{O} \cup \{P^{-}(a, b)\},\\
\mathcal{O} \cup \{\mathsf{Dis}(S, P),\ P(a, b)\} \text{ is a conservative extension of } \mathcal{O} \cup \{\neg S(a, b)\}
\end{gathered}
$$

where $\mathcal{O}$ is an arbitrary ontology, A is a fresh atomic concept and P is a fresh atomic role. Under this restriction, a finite set of data assertions becomes a dataset.

3.1.2 Translation to Rules

For convenience, we restrict ourselves to ontologies in a particular normal form.

Definition 3.1. An ontology axiom is *normalised* if it is in one of the forms in the left-hand-side of Table 3.2[2]. An ontology is *normalised* if it consists of normalised axioms only.

[2] We omit axioms of the form $A \sqsubseteq \geq n\, R.B$ as they can be simulated by $A \sqsubseteq \exists R.B_i$, $B_i \sqsubseteq B$ and $B_i \sqcap B_j \sqsubseteq \bot$ for $1 \leq i < j \leq n$ where each B_i is a fresh concept.

	Axioms	Rules
(O1)	$\prod_{i=1}^{m} A_i \sqsubseteq C_\bot$	$\bigwedge_{i=1}^{m} A_i(x) \to \bot$
(O2)	$\prod_{i=1}^{m} A_i \sqsubseteq \bigsqcup_{j=1}^{\ell} B_j$	$\bigwedge_{i=1}^{m} A_i(x) \to \bigvee_{j=1}^{\ell} B_j(x)$
(O3)	$\exists R.B \sqsubseteq A$	$\mathsf{ar}(R,x,y) \wedge B(y) \to A(x)$
(O4)	$\exists R.\mathsf{Self} \sqsubseteq A$	$\mathsf{ar}(R,x,x) \to A(x)$
(O5)	$A \sqsubseteq \exists R.B$	$A(x) \to \exists y(\mathsf{ar}(R,x,y) \wedge B(y))$
(O6)	$A \sqsubseteq \exists R.\mathsf{Self}$	$A(x) \to \mathsf{ar}(R,x,x)$
*(O7)	$A \sqsubseteq \forall R.B$	$A(x) \wedge \mathsf{ar}(R,x,y) \to B(y)$
*(O8)	$A \sqsubseteq \leq n\, R.B$	$A(x) \wedge \bigwedge_{i=1}^{n+1} \mathsf{ar}(R,x,y_i) \wedge B(y_i) \to \bigvee_{1 \leq i < j \leq n+1} y_i \approx y_j$
(O9)	$A \sqsubseteq \{a\}$	$A(x) \to x \approx a$
(O10)	$R \sqsubseteq S$	$\mathsf{ar}(R,x,y) \to \mathsf{ar}(R,x,y)$
(O11)	$R \circ S \sqsubseteq T$	$\mathsf{ar}(R,x,z) \wedge \mathsf{ar}(S,z,y) \to \mathsf{ar}(T,x,y)$
(O12)	$\mathsf{Dis}(R,S)$	$\mathsf{ar}(R,x,y) \wedge \mathsf{ar}(S,x,y) \to \bot$

where $n, m, \ell > 0$, A and B are atomic concepts, and R, S, T are roles.
*If A is $C_\top$ in (O7) and (O8), $A(x)$ can be eliminated from the body of the translation.

Table 3.2: Translation from normalised DL axioms to rules

Each normalised axiom corresponds to a single rule, as given on the right-hand-side of Table 3.2. Concept $C_\bot$ is translated as the specially nullary predicate $\bot$, whereas $C_\top$ is translated as an ordinary unary predicate, the meaning of which is axiomatised. Let π be the function that maps a normalised ontology axiom α to its corresponding rule in Table 3.2, and let $\mathcal{O}$ be a normalised OWL 2 ontology. Then, $\pi(\mathcal{O})$ is the smallest knowledge base containing:

- $\pi(\alpha)$ for each $\alpha \in \mathcal{O}$;
- a rule $A(x) \to C_\top(x)$ for each atomic concept A in $\mathcal{O}$; and
- rules $R(x,y) \to C_\top(x)$ and $R(x,y) \to C_\top(y)$ for each atomic role R in $\mathcal{O}$.

Note that since $\pi(\mathcal{O})$ is a knowledge base, it must contain equality axiomatisation of equality for its signature whenever $\approx$ is required to translate an axiom in $\mathcal{O}$. The translation from ontology axioms to rules is entailment-preserving; that is, $\mathcal{O} \cup \mathcal{D} \models \varphi$ iff $\pi(\mathcal{O}) \cup \mathcal{D} \models \varphi$ for any first-order sentence φ.

Note that the translation of an axiom α of the type (O5) does not lead to a normalised rule; however, $\pi(\alpha)$ can be seamlessly brought into normal form using the transformation in Proposition 2.1 by introducing a fresh binary predicate R_B.

$$A(x) \to \exists y\, R_B(x,y) \qquad R(x,y) \wedge B(y) \to R_B(x,y)$$
$$R_B(x,y) \to R(x,y) \qquad R_B(x,y) \to B(y)$$

Proposition 3.2. *An arbitrary OWL 2 ontology can be normalised into a set of axioms of the forms in the left-hand-side of Table 3.2 in polynomial time.*

Proof Sketch. Normalisation of ontologies can be seen as a variant of the well-known structure transformation (Nonnengart and Weidenbach, 2001). Each GCI in $\mathcal{O}$ can be brought into a normal form in polynomial time

$$\prod_{i=1}^{m} C_i \sqsubseteq \bigsqcup_{j=1}^{\ell} D_j, \tag{3.1}$$

where each C_i and D_i are formed according to the following grammar with A an atomic concept, R a role and n a positive integer (Motik et al., 2009c).

$$C \longrightarrow A \mid \exists R.A \mid \exists R.\mathsf{Self},$$
$$D \longrightarrow A \mid C_\bot \mid \{a\} \mid \exists R.A \mid \exists R.\mathsf{Self} \mid \forall R.A \mid \leq n\, R.B \mid \geq n\, R.B.$$

A GCI γ of the normal form (3.1) can be straightforwardly translated into axioms of the type (O1)–(O9) given in Table 3.2.

$$\Delta(\gamma) = \begin{cases} \gamma & \text{if } \gamma \text{ is of a form in Table 3.2,} \\ \{\prod_{i=1}^{m} A_{C_i} \sqsubseteq \bigsqcup_{j=1}^{\ell} A_{D_j}, C_i \sqsubseteq A_{C_i}, A_{D_j} \sqsubseteq D_j\} & \text{otherwise} \end{cases}$$

where A_C is C itself if C is an atomic concept, otherwise, a fresh atomic concept A_C that is uniquely associated with the complex concept C.

Each role axiom or role inclusion can be normalised into axioms of the type (O1), (O4), (O6) or (O10)–(O12) in Table 3.2 as follows:

$$\begin{aligned} \Delta(\mathsf{Ref}(R)) &= \{C_\top \sqsubseteq \exists R.\mathsf{Self}\}, && \text{O6} \\ \Delta(\mathsf{Irr}(R)) &= \{\exists R.\mathsf{Self} \sqsubseteq A_R, A_R \sqsubseteq C_\bot\}, && \text{O4, O1} \\ \Delta(\mathsf{Sym}(R)) &= \{R \sqsubseteq \mathsf{inv}(R)\}, && \text{O10} \\ \Delta(\mathsf{Asy}(R)) &= \{\mathsf{Dis}(R, \mathsf{inv}(R))\}, && \text{O12} \\ \Delta(\mathsf{Tra}(R)) &= \{R \circ R \sqsubseteq R\}, && \text{O11} \\ \Delta(R_1 \circ \omega \sqsubseteq R) &= \Delta(R_1 \circ P_\omega \sqsubseteq R) \cup \Delta(\omega \sqsubseteq P_\omega) && \text{O11} \end{aligned}$$

where A_R is a fresh atomic concept uniquely associated to a role R, and P_ω is a fresh atomic role that is unique for the role composition ω.

Given an arbitrary OWL 2 ontology consisting of GCIs of the form (3.1), role axioms and role inclusions, $\Delta(\mathcal{O}) = \bigcup_{\gamma \in \mathcal{O}} \Delta(\gamma)$ consists of normalised axioms and $\Delta(\mathcal{O})$ can be computed in polynomial time. □

3.1.3 Inference Problems

Traditional reasoning tasks in DLs include satisfiability checking, concept subsumption tests and instance retrieval. Let $\mathcal{O}$ be an ontology, let $\mathcal{D}$ be a set of data assertions, and let C, D be two concepts in the signature of $\mathcal{O}$.

Satisfiability checking is to verify if there exists a model of $\mathcal{O} \cup \mathcal{D}$ when given $\mathcal{O}$ and $\mathcal{D}$ as input. Satisfiability is the prerequisite of any other reasoning task. If $\mathcal{O} \cup \mathcal{D}$ is unsatisfiable, it trivially entails every axiom and assertion in its signature. *Concept subsumption tests* are usually performed w.r.t. an ontology only. The concept subsumption test in $\mathcal{O}$ for $C \sqsubseteq D$ determine if the extension of C is a subset of the extension of D in all models of $\mathcal{O}$. This task is crucial to construct a taxonomy (a.k.a. concept hierarchy) in the domain of interest. In DL terminologies, the task to compute all concept subsumptions of atomic concepts is called *classification* over an ontology. In particular, the subsumption test $C \sqsubseteq C_\bot$ for a concept C also determines the *satisfiability* of C in a ontology. If $C \sqsubseteq C_\bot$ is entailed, then C is *unsatisfiable* in the ontology, otherwise, *satisfiable*. *Instance problem* of concept C is to compute the instances of C in all models of $\mathcal{O} \cup \mathcal{D}$.

In addition to the aforementioned traditional reasoning tasks above, querying answering plays a more and more important role in data-intensive applications. The instance checking problem of atomic concepts corresponds a basic kind of queries — instance queries. Different query languages with specific semantics are available to form more complex and realistic queries (see details in Chapter 4). Moreover, *entailment of data assertions* is to determine if $\mathcal{O} \cup \mathcal{D} \models \alpha$ for a data assertion α.

Subsumption tests can be reduced to instance problem as follows. Let a_C be a fresh individual that does not appear in $\mathcal{O}$,

$$\mathcal{O} \models C \sqsubseteq D \text{ iff } \mathcal{O} \cup \{C(a_C)\} \models D(a_C).$$

Therefore, classification can be reduced to a set of instance problems.

In most DLs, instance problem and entailment of data assertions can be reduced to satisfiability checking in the following sense. For example, for any constant a,

$$\mathcal{O} \cup \mathcal{D} \models C(a) \text{ iff } \mathcal{O} \cup \mathcal{D} \cup \{\bar{C}(a),\ C \sqcap \bar{C} \sqsubseteq C_\bot\} \text{ is unsatisfiable.}$$

These basic inference problems are of very high computational complexity for OWL-DL and OWL 2. Satisfiability checking is NExpTime for $\mathcal{SHOIQ}$ (Tobies, 2000) and 2NExpTime for $\mathcal{SROIQ}$ (Kazakov, 2008). This leads the development of logic fragments that are computationally easier.

3.2 Logic Fragments

The OWL 2 standard addresses its high computational complexity by introducing trimmed down profiles that "trade some expressive power for the efficiency of reasoning", including OWL 2 QL profile, OWL 2 EL profile, and OWL 2 RL profile. All profiles are Horn, where we say a normalised ontology is *Horn* if every axiom of type (O2) satisfies $\ell = 1$ and every axiom of type (O8) satisfies $n = 1$ in Table 3.2.

3.2.1 OWL 2 QL

OWL 2 QL profile is defined based on a family of lightweight DLs, called *DL-Lite*. *DL-Lite* is specially tailored to capture basic ontology languages without sacrificing scalability for reasoning tasks. In addition to supporting standard reasoning services, *DL-Lite* is designed for applications over large datasets where query answering is the most important reasoning task (Calvanese et al., 2007). There are different variants in the *DL-Lite* family, among which *DL-Lite*$_{\mathcal{R}}$ provides the logical underpinning for OWL 2 QL. In OWL 2 QL, concepts are defined by the following syntax:

$$C \longrightarrow A \mid C_{\bot} \mid \exists R$$

where A is an atomic concept, R is a role. A GCI in OWL 2 QL is of the form $C \sqsubseteq D$ for OWL 2 QL concepts C and D; complex role inclusions are not allowed in OWL 2 QL; and role axioms allowed in OWL 2 QL are restricted to $\mathsf{Ref}(R)$, $\mathsf{Sym}(R)$, $\mathsf{Asy}(R)$, and $\mathsf{Dis}(R, S)$ with roles R, S. Please note that OWL 2 QL captures qualified existential restrictions in the right-hand-side of GCI implicitly through an interaction between role inclusions and inverse roles, e.g. an axiom in the form of $A \sqsubseteq \exists R.B$ can be captured by $\{A \sqsubseteq \exists R_B,\ R_B \sqsubseteq R,\ \exists R_B^{-} \sqsubseteq B\}$ with R_B a new predicate.

Proposition 3.3. *An arbitrary OWL 2 QL ontology can be normalised into a set of axioms of the type (O1)–(O6), (O10), and (O12) in Table 3.2.*

A normalised Horn ontology is in OWL 2 QL if it consists of axioms of type (O1)–(O6), (O10), and (O12) where every axiom of type (O3) satisfies $B = C_{\top}$, every axiom of type (O4) satisfies $A \sqsubseteq C_{\bot}$, and every axiom of type (O6) satisfies $A = C_{\top}$.

The language *DL-Lite*$_{core}$ is the basis of the whole family, which can be obtained by disallowing role inclusions and role axioms in *DL-Lite*$_{\mathcal{R}}$. Other variants of *DL-Lite* have also been described in the literature, e.g. *DL-Lite*$_{\mathcal{A}}$ (Poggi et al., 2008) and *DL-Lite*$_{horn}$ (Artale et al., 2009).

In OWL 2 QL, polynomial time algorithms exist for satisfiability checking and concept subsumption tests; and query answering can be done in LogSpace (actually in AC^0) w.r.t. data complexity (Calvanese et al., 2007).

3.2.2 OWL 2 EL

OWL 2 EL is based on the $\mathcal{EL}$ family of DLs. In OWL 2 EL, concepts are formed according to the following syntax:

$$C, D \longrightarrow A \mid C_\bot \mid \{a\} \mid C \sqcap D \mid \exists P.C \mid \exists P.\mathsf{Self}$$

where A denotes an atomic concept, and P an atomic role. A GCI in OWL 2 EL is of the form $C \sqsubseteq D$ for OWL 2 EL concepts C and D; roles are restricted to atomic roles; (complex) role inclusions are allowed; and role axioms are restricted to $\mathsf{Ref}(R)$ and $\mathsf{Tra}(R)$. Additionally, range restrictions of the form $C_\top \sqsubseteq \forall R.A$ are allowed in OWL 2 EL.

Proposition 3.4. *An arbitrary OWL 2 EL ontology can be normalised into a set of axioms of the type (O1)—(O7) and (O9)—(O11) in Table 3.2.*

We say that a normalised Horn ontology is in OWL 2 EL if it consists of axioms of type (O1)—(O7) and (O9)—(O11), where all roles occurring in the ontology are restricted to atomic roles, every axiom of type (O7) satisfies $A = C_\top$, and every axiom of type (O11) satisfies $R = S = T$.

A fragment of OWL 2 EL, called $\mathcal{ELHO}^r_\bot$, will be exploited as a light-weight ontology language to compute a lower bound of query answers (see Chapter 7), which extends $\mathcal{ELH}$ with the bottom concept, nominals, and range restrictions. We say that a normalised Horn ontology is in $\mathcal{ELHO}^r_\bot$ if it does not contain axioms of the type (O4), (O6), (O8), (O11), or (O12), every axiom of type (O7) satisfies $A = C_\top$, and roles occurring in any axioms are restricted to atomic roles. Rules corresponding to $\mathcal{ELHO}^r_\bot$ axioms can be syntactically characterised as follows.

Definition 3.5. A rule is in $\mathcal{ELHO}^r_\bot$ if it is of one of the following forms:

$$\bigwedge_{i=1}^{p} A_i(x) \wedge \bigwedge_{j=1}^{q} [R_j(x, y_j) \wedge \bigwedge_{k=1}^{l_j} B_{jk}(y_j)] \to \varphi(x), \tag{EL1}$$

$$R_1(x, y) \to R_2(x, y), \tag{EL2}$$

$$R(x, y) \to A(y). \tag{EL3}$$

where $\varphi(x)$ is either $\bot$, or of the form $A(x)$, $x \approx c$, or $\exists y R(x, y)$

3.2.3 OWL 2 RL

Description Logic Programs (DLP) (Grosof et al., 2003) forms the logic foundation of OWL 2 RL, which corresponds to the "intersection" between DL and datalog. So query evaluation in OWL 2 RL is PTIME-complete w.r.t. the size of data (Abiteboul et al., 1995). Sub-concepts C and super-concepts D in OWL 2 RL are formed according to the following syntax.

$$C \longrightarrow B \mid \{a\} \mid C_1 \sqcap C_2 \mid \exists R.C.$$
$$D \longrightarrow B \mid C_\bot \mid \neg C \mid \forall R.D \mid \exists R.\{a\} \mid \leq 1\, R.C.$$

where B is an atomic concept other than $C_\top$, each $C_{[i]}$ a sub-concept and R a role.

Proposition 3.6. *An arbitrary OWL 2 RL ontology can be normalised into a set of axioms of the type (O1)–(O3), (O7), (O8), (O10)–(O12) in Table 3.2.*

We say that a normalised Horn ontology is in OWL 2 RL if it contains axioms of type (O1)–(O3), (O7), (O8), and (O10)–(O12) where every axioms of type (O2) the left-hand-side concept of any axiom in the ontology is not $C_\top$.

As discussed in the introduction, although OWL 2 RL captures a substantial fragment of OWL 2, it doesn't allow disjunctions and existential quantifiers in the right-hand-side of a GCI. Moreover, though OWL 2 RL is closely related to datalog, there are several kinds of OWL 2 axioms that are captured by datalog, but are not included in the RL profile. For instance, axioms satisfying one of the following conditions is not in OWL 2 RL.

- All GCIs that the Self constructor is involved, like $C \sqsubseteq \exists R.\mathsf{Self}$ and $\exists R.\mathsf{Self} \sqsubseteq D$, as well as reflexivity of roles;
- GCIs of the form $C \sqsubseteq \{a\}$; and
- GCIs with $C_\top$ as the left-hand-side concept.

Comparison between Profiles Traditional reasoning tasks for the three OWL 2 profiles, QL, EL and RL, are in polynomial time w.r.t. the size of an input ontology and a set of individual assertions. Query answering is also in PTIME to the number of input data assertions.

All three profiles in OWL 2 are Horn, so disjunctive knowledge is not captured. For instance, the OWL 2 axiom $\mathsf{Mammal} \sqsubseteq \mathsf{Herbivore} \sqcup \mathsf{MeatEater}$, that is equivalent

		OWL 2 QL	OWL 2 EL	OWL 2 RL
$(A1)$	$\exists \mathsf{eats}.\mathsf{Herbivore} \sqsubseteq \mathsf{MeatEater}$	$\times$	✓	✓
$(A2)$	$\mathsf{tra}(\mathsf{isSameGenus})$	$\times$	✓	✓
$(A3)$	$\mathsf{Mammal} \sqsubseteq \exists \mathsf{eats}$	✓	✓	$\times$
$(A4)$	$\mathsf{Herbivore} \sqsubseteq \forall \mathsf{eats}.\mathsf{Plant}$	$\times$	$\times$	✓
$(A5)$	$\mathsf{Habitat} \sqsubseteq \exists \mathsf{growsIn}^-$	✓	$\times$	$\times$

Table 3.3: Comparison between OWL 2 profiles

to rule (1.1) in the introduction, is not expressible in any of the profiles. On the other hand, the expressivity of different profiles is incomparable. Table 3.3 gives several example axioms that distinguish the expressivity of three OWL 2 profiles.

- OWL 2 QL support very limited kinds of role axioms, and quantified existential concepts are note allowed in the left-hand-side of a GCI.
- OWL 2 EL doesn't capture $(A4)$ since the left-hand-side concept in an axiom of type (O7) is restricted to $C_\top$; it doesn't capture $(A5)$ since inverse roles are not allowed.
- OWL 2 RL doesn't capture $(A3)$ since existential quantifiers are not allowed in the right-hand-side of a GCI.

Chapter 4

Query Languages for Ontologies

An important aspect of semantic web technologies is the ability to retrieve answers to queries. To achieve this goal, different query languages have been developed. We next introduce two kinds of widely used query languages in ontologies—conjunctive queries in Section 4.1 and SPARQL queries in Section 4.2. They are also closely related to each other.

4.1 Conjunctive Queries

A *conjunctive query* is a function-free first-order formula $q(\mathbf{x}) = \exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$ where $\varphi(\mathbf{x}, \mathbf{y})$ is a conjunction of equality-free atoms over variables $\mathbf{x} \cup \mathbf{y}$. A query is

- *ground* if $|\mathbf{y}| = 0$;
- *atomic* if it is ground and φ consists of a single atom;
- *Boolean* if $|\mathbf{x}| = 0$; or
- an *instance query* if it is an atomic query over a unary predicate.

For simplicity, we sometimes omit free variables in queries and write q instead of $q(\mathbf{x})$. Furthermore, we treat conjunctions of atoms in queries as sets and hence we do not allow for duplicated atoms in a query. The sets of free variables, existentially quantified variables, and constants in q are denoted by $\mathsf{f\text{-}var}(q)$, $\mathsf{e\text{-}var}(q)$ and $\mathsf{const}(q)$, respectively.[1] By slight abuse of notation, we sometimes write $\alpha \in q$ for an atom α if α occurs in q.

[1] Please note that $\mathsf{f\text{-}var}(q)$ and $\mathsf{e\text{-}var}(q)$ may be referred as distinguished and non-distinguished variables, respectively, in other literature.

Example 4.1. The query $q_{ex}(x) = \exists y[\mathsf{eats}(x, y) \wedge \mathsf{Plant}(y)]$ is a conjunctive query that asks for all objects that eat a plant. ♢

Let $\mathcal{K}$ be a knowledge base. A substitution σ to constants is a *possible answer* to q w.r.t. $\mathcal{K}$, if $\mathsf{dom}(\sigma) = \mathbf{x}$. A possible answer is a *(certain) answer* to q w.r.t. $\mathcal{K}$ if $\mathcal{K} \models q\sigma$; the set of certain answers is denoted by $\mathsf{cert}(q, \mathcal{K})$. Note that, if $\varphi(\mathbf{x}, \mathbf{y})$ is Boolean, then the set of certain answers is either empty or it contains the empty substitution. We treat unsatisfiability as a special Boolean query where $\varphi(\mathbf{x}, \mathbf{y}) = \bot$, which holds w.r.t. $\mathcal{K}$ iff $\mathcal{K}$ is unsatisfiable. For convenience and by slight abuse of notation, in our examples, we also treat a tuple $\mathbf{a}$ of constants from $\mathcal{K}$ of the same arity as $\mathbf{x}$ as a possible answer and say that $\mathbf{a}$ is a certain answer, written as $\mathbf{a} \in \mathsf{cert}(q, \mathcal{K})$, if $\mathcal{K} \models q(\mathbf{a})$.

Conjunctive queries can be alternatively represented using single datalog rules. To this end, each query q is uniquely associated with a predicate P_q, where we take $P_\bot = \bot$ and a set $\mathcal{R}_q$ of rules as follows.

$$\mathcal{R}_q = \begin{cases} \emptyset & q = \bot, \\ \{\varphi(\mathbf{x}, \mathbf{y}) \to P_q(\mathbf{x})\} & \text{otherwise.} \end{cases} \tag{4.1}$$

In this way, certain answers can be characterised by means of entailment of single facts. Then, $\sigma \in \mathsf{cert}(q, \mathcal{K})$ iff $\mathcal{K} \cup \mathcal{R}_q \models P_q(\mathbf{x})\sigma$.

Answering conjunctive queries w.r.t. knowledge bases is computationally very hard, and decidability for knowledge bases stemming from OWL 2 remains open. A modification of the first-order semantics, called the *ground semantics*, however, can be exploited to regain decidability. The *ground semantics* that is adopted by the standard query language SPARQL 1.1 (W3C SPARQL Working Group, 2013) can be formulated as follows: a possible answer σ is a *ground answer* to $q(\mathbf{x}) = \exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$ w.r.t. a knowledge base $\mathcal{K}$ if there exists a substitution σ' from $\mathbf{y}$ to constants of $\mathcal{K}$ such that $\mathcal{K} \models \varphi\sigma\sigma'$. Clearly, every ground answer is a certain answer, but not vice versa. The set of ground answers to q w.r.t. $\mathcal{K}$ is denoted by $\mathsf{ground}(q, \mathcal{K})$.

It is easy to see that the first-order semantics of q w.r.t. $\mathcal{K}$ is equivalent to the ground semantics in the following situations:

- $\mathcal{K}$ is a datalog knowledge base, or
- q is a ground conjunctive query.

In general, however, there is a discrepancy between both semantics. This is the case, for example, if $\mathcal{K}$ contains existentially quantified rules.

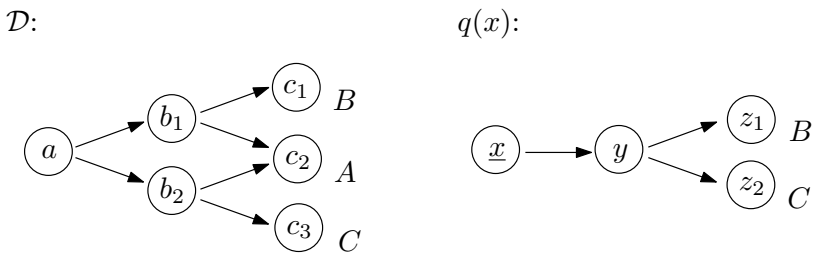


Figure 4.1: The dataset and query for Example 4.3

Example 4.2. Consider a set Σ consisting of a single rule $\mathsf{Mammal}(x) \rightarrow \exists y\, \mathsf{eats}(x, y)$, a dataset $\mathcal{D} = \{\mathsf{Mammal}(\mathit{tiger}), \mathsf{eats}(\mathit{python}, \mathit{rabbit})\}$ and a query $q(x) = \exists y\, \mathsf{eats}(x, y)$. We have $\mathsf{cert}(q, \Sigma \cup \mathcal{D}) = \{\mathit{tiger}, \mathit{python}\}$, but $\mathsf{ground}(q, \Sigma \cup \mathcal{D}) = \{\mathit{python}\}$. Since $\mathsf{eats}(\mathit{python}, \mathit{rabbit})$ is explicit in the dataset, we have that *python* is a certain answer and a ground answer to q. The certain and ground answers to q, however, differ on the constant *tiger*. Since *tiger* is an instance of Mammal, we have that *tiger* is connected in every model of $\Sigma \cup \mathcal{D}$ via the eats relation to some domain element. Thus, the existentially quantified variable y in q can always be matched to such a domain element in every model. We have that *tiger* is a certain answer to q. ♢

Rules with disjunctions in the head in $\mathcal{K}$ may also lead to different query answers under the first-order and ground semantics.

Example 4.3. Consider a set Σ consisting of a single rule $A(x) \rightarrow B(x) \vee C(x)$, a dataset $\mathcal{D}$ depicted in Figure 4.1 and a query

$$q(x) = \exists y \exists z_1 \exists z_2 [R(x, y) \wedge R(y, z_1) \wedge B(z_1) \wedge R(y, z_2) \wedge C(z_2)].$$

We have that $\mathsf{cert}(q, \Sigma \cup \mathcal{D}) = \{a\}$, but $\mathsf{ground}(q, \Sigma \cup \mathcal{D}) = \emptyset$. From the fact $A(c_2)$ and the rule in Σ, we know $B(c_2)$ or $C(c_2)$ holds. If $C(c_2)$ holds, the substitution $\sigma_1 = \{x \mapsto a, y \mapsto b, z_1 \mapsto c_1, z_2 \mapsto c_2\}$ maps q the a subset of $\mathcal{D}$, and hence $q(a)$ holds in this case. Otherwise, if $B(c_2)$ holds, $\sigma_2 = \{x \mapsto a, y \mapsto b, z_1 \mapsto c_2, z_2 \mapsto c_3\}$ maps q to a different subset of $\mathcal{D}$. So $a \in \mathsf{cert}(q, \Sigma \cup \mathcal{D})$. However, neither $q\sigma_1$ or $q\sigma_2$ is entailed by $\Sigma \cup \mathcal{D}$. Therefore, the ground answers to q is empty. ♢

In OWL 2 ontologies, predicates are restricted to be either unary (for atomic concepts) or binary (for atomic roles). Therefore, conjunctive queries for ontologies are restricted to mention only unary predicates and/or binary predicates in atoms. For the reminder of this chapter, we restrict ourselves to such queries.

As already mentioned, the decidability of conjunctive query answering over OWL 2 ontologies is still open. We next recapitulate classes of queries, for which query

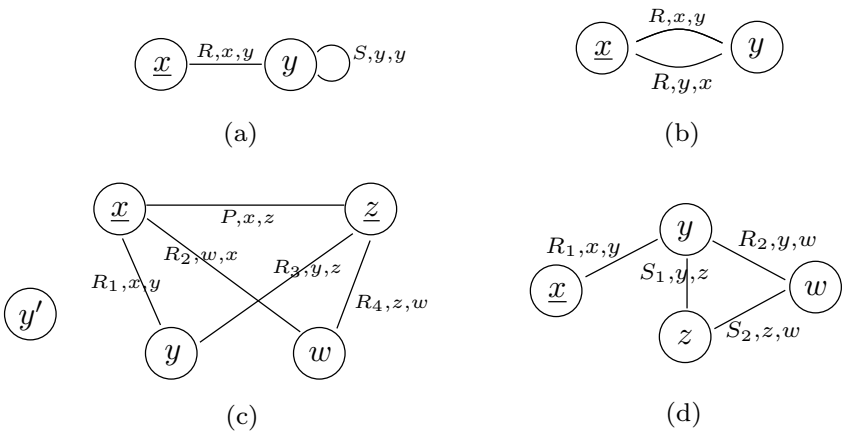


Figure 4.2: Term graphs of queries in Example 4.5

answering is decidable over OWL 2. To define these classes of queries, we use an undirected graph representation of queries.

Definition 4.4. Let q be a conjunctive query. The *term graph* $\mathcal{U}(q) = \langle V, E \rangle$ is the undirected multi-graph defined as follows:

- V is the set of all terms in q, and
- E is the set of undirected edges $e_{P,u,v}$ connecting terms u and v for each atom $P(u, v) \in q$. ♢

Note that $\mathcal{U}(q)$ is a multi-graph, and hence self-loops and multiple edges between two terms are allowed. Furthermore, each edge is unique for a triple (P, u, v); thus, if a query contains atoms $P(u, v)$ and $P(v, u)$, its term graph contains two different edges for (P, u, v) and (P, v, u).

Example 4.5. Figure 4.2 gives the term graphs of the following queries, where free variables are underlined.

(a) $q(x) = \exists y[R(x, y) \wedge S(y, y)]$

(b) $q(x) = \exists y[R(x, y) \wedge R(y, x)]$

(c) $q(x, z) = \exists y \exists w \exists y'[P(x, z) \wedge R_1(x, y) \wedge R_2(w, x) \wedge R_3(y, z) \wedge R_4(z, w) \wedge A(y')]$

(d) $q(x) = \exists y \exists z \exists w[R_1(x, y) \wedge S_1(y, z) \wedge R_2(y, w) \wedge S_2(z, w)]$

A self-loop occurs in Figure 4.2a for the atom $S(y, y)$, and there are two edges between x, y in Figure 4.2b labeled with different triples for $R(x, y)$ and $R(y, x)$. ♢

Tree- and Forest-shaped Queries Tree- and forest-shaped queries are well-known classes of queries that are supported by OWL 2 reasoners. Tree-shaped queries are especially important in DLs, as they can be transformed into single concepts.

Definition 4.6. The query q is *forest-shaped* if the subgraph obtained from $\mathcal{U}(q)$ by removing all edges involving only free variables is a forest rooted at $\mathsf{f\text{-}var}(q)$. A forest-shaped query is *tree-shaped* if it contains a single free variable. ♢

The DL concept that corresponds to a tree-shaped query can be defined inductively as follows based on the parent function of its term graph (deemed as a tree rooted at the only free variable).

Definition 4.7. Let q be a tree-shaped query, let x be the only free variable in q, and let $\mathsf{pa}(\cdot)$ be the parent function of $\mathcal{U}(q)$ rooted at x. The rolled-up concept C_u for each term u in $\mathcal{U}(q)$ is defined bottom-up on the structure of the rooted tree.

$$C_u = \bigsqcap_{C(u) \in q} C \sqcap \bigsqcap_{\substack{u = \mathsf{pa}(v), \\ \mathsf{ar}(R,u,v) \in q, \\ v \in \mathsf{const}(q)}} \exists R.(\{v\} \sqcap C_v) \sqcap \bigsqcap_{\substack{u = \mathsf{pa}(v), \\ \mathsf{ar}(R,u,v) \in q, \\ v \in \mathsf{e\text{-}var}(q)}} \exists R.C_v.$$

Finally, the rolled-up concept C_q of q is defined as C_x. ♢

Let $\mathcal{L}$ be a DL language. A tree-shaped query is said to be in $\mathcal{L}$ if the rolled-up concept of q is an $\mathcal{L}$-concept. Answering tree-shaped queries in $\mathcal{L}$ over $\mathcal{L}$-ontologies can be reduced to instance queries.

Theorem 4.8. *Let $\mathcal{K}$ be a knowledge base. For any tree-shaped query q and any constant a, we have $a \in \mathsf{cert}(\mathcal{K}, q)$ iff $\mathcal{K} \models C_q(a)$.*

Proof. Let $\mathcal{I}$ be a model of $\mathcal{K}$. We next prove by induction on the depth of the tree corresponding to q that $\mathcal{I} \models q(a)$ iff $\mathcal{K} \models C_x(a)$ for any tree-shaped query q. In this proof, we simply use the depth of a tree-shaped query q to refer the depth of the tree $\mathcal{U}(q)$ rooted at the free variable.

Base case: The tree corresponding to q is of depth zero; that is, the tree contains only the root x. Assume that $q(x) = A_1(x) \wedge \cdots \wedge A_n(x)$ then $C_x = A_1 \sqcap \cdots \sqcap A_n$. So by the semantics of query answers and the semantics of DLs, $\mathcal{I} \models q(a)$ iff $\mathcal{I} \models C_x(a)$.

Inductive step: The induction hypothesis is that $\mathcal{I} \models q'(a)$ iff $\mathcal{I} \models C_{q'}(a)$ for any query q of depth less than k. We next prove that this claim holds for q of depth k. Let q be a tree-shaped query of depth k with the free variable x, and let $v_1, \ldots, v_m$ be the children of x in $\mathcal{U}(q)$. So each sub-tree rooted at v_i in $\mathcal{U}(q)$

corresponds to a tree-shaped query q_i of depth less than k. Then q is of the form $q(x) = \exists y_1 \dots \exists y_m \bigwedge_{i=1}^{m} (R_i(x, y_i) \wedge q_i(y_i))$. Then $C_x = \exists R_1.C_1 \wedge \dots \wedge \exists R_m.C_m$. Let C_i be the rolled-up concept of q_i for each $1 \leq i \leq m$. By induction hypothesis, we have $\mathcal{I} \models q_i(a)$ iff $\mathcal{I} \models C_i(a)$ for any constant a. So by the semantics of query answers and the semantics of DLs, $\mathcal{I} \models q(a)$ iff $\mathcal{I} \models C_x(a)$. □

Example 4.9. The query $q(x) = \exists y[R(x, y) \wedge R(y, x)]$ depicted in Figure 4.2b is not tree-shaped since its term graph contains a cycle. The query q_{ex} in Example 4.1 and $q(y) = \exists x[\mathsf{eats}(x, y) \wedge \mathsf{Plant}(y)]$ are both tree-shaped, and the rolled-up concepts of them are $\exists \mathsf{eats}.\mathsf{Plant}$ and $\exists \mathsf{eats}^- \sqcap \mathsf{Plant}$, respectively. Considering the $\mathcal{ELHO}^r_\bot$ fragment of OWL 2, we have only q_{ex} is in $\mathcal{ELHO}^r_\bot$ since $\exists \mathsf{eats}.\mathsf{Plant}$ is a $\mathcal{ELHO}^r_\bot$-concept but $\exists \mathsf{eats}^- \sqcap \mathsf{Plant}$ is not. ♢

Internalisable Queries We next define the class of internalisable queries, which extends tree- and forest-shaped queries. Answering internalisable queries in OWL 2 ontologies can be reduced to entailment of data assertions as well as to satisfiability, which implies decidability of the problem. We discuss the reduction in detail in Section 10.2.

Definition 4.10. A conjunctive query q is *internalisable* if the term graph $\mathcal{U}(q)$ does not contain a cycle of length at least two involving only variables from $\mathsf{e\text{-}var}(q)$. ♢

Any ground query q is internalisable as $\mathsf{e\text{-}var}(q) = \emptyset$. Similarly, any instance query is internalisable. Moreover, every forest-shaped query q is also internalisable, as it contains no cycles.

Example 4.11. None of the queries in Example 4.5 is forest-shaped. However, all but that in Figure 4.2d are internalisable. Although (a), (b), (c) contain cycles in their term graph, they are still internalisable since each of the cycles involves some free variables. The query in (d) is not internalisable, as it contains a cycle y, z, w involving only existentially quantified variables. ♢

We conclude this section by introducing two extensions of conjunctive queries—unions of conjunctive queries and datalog queries. They are also common target languages for rewriting techniques to answer conjunctive queries over ontologies in light-weight DLs (see Section 5.2.3).

Union of Conjunctive Queries A *union of conjunctive queries (UCQ)* is a query of the form $q(\mathbf{x}) = \bigvee_{i=1}^{m} \exists y_i \varphi_i(\mathbf{x}, \mathbf{y}_i)$ where each $\exists \mathbf{y}_i \varphi_i(\mathbf{x}, \mathbf{y}_i)$ is a conjunctive query. Thus, a conjunctive query is a UCQ consisting of a single disjunct.

Datalog Queries In addition to an ontology language, datalog can also be seen as a query language that extends UCQs with recursion (Abiteboul et al., 1995). A *datalog query* q is of the form $\langle P_q, \Sigma \rangle$ where Σ is a set of datalog rules and P_q is a unique predicate for the query q. As mentioned before, a conjunctive query $q_1 = \exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$ can be alternatively represented by the datalog query $\langle P_{q_1}, \{\varphi(\mathbf{x}, \mathbf{y}) \rightarrow P_{q_1}(\mathbf{x})\} \rangle$. Similarly, a UCQ $q_2 = \bigvee_{i=1}^{m} \exists \mathbf{y}_i \varphi_i(\mathbf{x}, \mathbf{y}_i)$ can be expressed by the datalog query $\langle P_{q_2}, \{\varphi_i(\mathbf{x}, \mathbf{y}) \rightarrow P_{q_2}(\mathbf{x}) \mid 1 \leq i \leq m\} \rangle$. A datalog query q is *linear* if for each rule in q, there is at most one body atom whose predicate is mutually recursive with the predicate in the head.

4.2 SPARQL

SPARQL (W3C SPARQL Working Group, 2013) is a W3C standard query language to retrieve and manipulate data stored in RDF format, in the same way as SQL is standardised for relational databases.

RDF Graphs

The Resource Description Framework (RDF) is a widely used data model to represent description of web resources by means of simple statements.

The signature of an RDF graph consists of pair-wise disjoint sets of *International Resource Identifiers* (IRI) I, *RDF literals* L, and *blank nodes* (syntactically specified by $\_:name$) B. The set T of RDF terms is defined as $I \cup L \cup B$. A triple of the form $(s\ p\ o)$ from $(I \cup B) \times I \times T$ is called an *RDF triple*, where s is called the *subject*, p the *predicate*, and o the *object*. An *RDF graph* is a finite set of RDF triples.

We generally abbreviate IRIs using prefixes rdf:, rdfs: and owl: to refer to the RDF, RDFS and OWL namespaces, respectively. Moreover, we fix a prefix : for a default namespace. An RDF graph G corresponds to a directed labeled graph consisting of edges from s to o labeled with p for each triple $(s\ p\ o)$ in G.

Syntax and Semantics

The basic building blocks of a SPARQL query are *triple patterns* of the form $(s\ p\ o) \in (I \cup B \cup V) \times (I \cup V) \times (T \cup V)$ where V is a set of query variables (syntactically specified by $?x$ for a variable x) disjoint with the set T of RDF terms. A *basic graph pattern* (BGP) is a finite set of triple patterns.

Any function-free unary and binary atom in first-order logic can be mapped into a triple pattern. In the mapping between triple patterns and first-order atoms, IRIs are used to identify first-order predicates and some of first-order constants; RDF literals correspond to first-order constants; query variables and blank nodes are mapped to first-order variables. For convenience, we map a query variable $?x$ to variable x and a blank node $\_{:}y$ to variable y. For example, ($?x$ rdf:type :Plant) is a triple pattern to represent the unary atom $\mathsf{Plant}(x)$, and ($?x$:eats $\_{:}y$) can be used to express the binary atom $\mathsf{eats}(x, y)$. Since variables are allowed in any predicate and object position, some triple patterns do not directly correspond to first-order atoms. For instance, ($sheep$ rdf:type $?x$) and ($sheep$ $?y$ $grass$), both of which can be seen as extended first-order atoms $x(sheep)$ and $y(sheep, grass)$ where variables are allowed in places of predicates.

A *SPARQL conjunctive query* is an expression of the form

$$\texttt{SELECT}\ ?x_1 \ldots ?x_n\ \texttt{WHERE}\ \{\ s_1\ p_1\ o_1\ .\quad \ldots\ s_m\ p_m\ o_m\ \}$$

where each query variable in $\{?x_1, \ldots, ?x_n\}$ appears a triple pattern $(s_i\ p_i\ o_i)$, and triple patterns in the `WHERE` clause are separated by a full stop. The BGP of this query is defined as $\{(s_1\ p_1\ o_1), \ldots, (s_m\ p_m\ o_m)\}$. The set of *answer variables* of this query is defined as $\{x_1, \ldots, x_n\}$.

Example 4.12. The conjunctive query q_{ex} in Example 4.1 can be represented by the following SPARQL conjunctive query

$$\texttt{SELECT}\ ?x\ \texttt{WHERE}\ \{\ ?x\ \text{:eats}\ ?y\ .\quad ?y\ \text{rdf:type}\ \text{:Plant}\ \}. \qquad \diamondsuit$$

The semantics of SPARQL conjunctive queries over RDF graphs is based on subgraph matching. Formally, a *solution* for a SPARQL conjunctive query q is a mapping σ from variables and blank nodes in q to nodes in the queried RDF graph G, such that the application of σ to the BGP of q results in a subgraph of G. *Answers* to a SPARQL conjunctive query are the projections on answer variables of all solutions. Please note that blank nodes play a similar role to variables in SPARQL queries with the main difference being that blank nodes cannot occur in the `SELECT` clause. Furthermore, blank nodes in SPARQL queries cannot refer to specific blank nodes in the queried graph.

OWL 2 Direct Semantics Entailment Regime

The semantics of SPARQL queries w.r.t. OWL 2 ontologies is specified by OWL 2 Direct Semantics entailment regime (Glimm and Ogbuji, 2012).

RDFS/OWL provides a vocabulary extension of RDF, so that ontology axioms and data assertions can be mapped into a set of RDF triples. For instance, the predicate rdfs:subClassOf in the reserved vocabulary of RDFS can be used to map the axiom Herbivore $\sqsubseteq$ Mammal to the triple (:Herbivore rdfs:subClassOf :Mammal). For further details, we refer interested readers to the W3C specification for the mapping between OWL structural objects to RDF triples (Motik et al., 2009b)

Syntax OWL 2 entailment regime is defined on a set of legal graphs that can be mapped into OWL 2 ontologies and a set of legal queries for each legal RDF graph. A BGP satisfies the typing constraints of the entailment regime if no variable is declared as being of more than one type in OWL.[2] A SPARQL conjunctive query is *legal* for a legal RDF graph G if it satisfies the typing constraints and its BGP can be mapped to an extended OWL 2 ontology where variables are allowed in any places of predicates or constants as specified in the declaration axioms in G.

Example 4.13. The following RDF graph G is a legal graph under OWL 2 entailment regime. For brevity, we ignore OWL declaration axioms in legal graphs.

$$\{(:rabbit\ \text{rdf:type}\ :\text{Herbivore}),\quad (:sheep\ :\text{eats}\ :grass),\quad (:grass\ \text{rdf:type}\ :grass),$$
$$(\text{Mammal}\ \text{rdfs:subClassOf}\ C),\quad (C\ \text{rdf:type}\ \text{owl:Restriction}),$$
$$(C\ \text{owl:onProperty}\ :\text{eats}),\quad (C\ \text{owl:someValuesFrom}\ :\text{Plant})\}$$

The RDF graph G corresponds to the following OWL 2 ontology $\mathcal{O}$.

$$\mathcal{O} = \{\text{Herbivore}(rabbit), \text{eats}(sheep, grass), \text{Mammal} \sqsubseteq \exists \text{eats.Plant}\}$$

Query Q1 is legal for G since its BGP corresponds to an extended ontology $\{\text{eats}(x, y), \text{Plant}(y)\}$ where variables are allows in places of constants.

$$\texttt{SELECT}\ ?x\ \texttt{WHERE}\ \{\ ?x\ :\text{eats}\ ?y\ .\quad ?y\ \text{rdf:type}\ :\text{Plant}\ \} \qquad \text{(Q1)}$$

[2] Types in OWL include owl:Class, owl:ObjectProperty, owl:DatatypeProperty, owl:NamedIndividual.

Similarly, query Q2 is legal for G as it can be mapped to an extended ontology $\{\exists \mathsf{eats}.\mathsf{Plant}(x)\}$ where variables are allowed in places of constants.

```
SELECT ?x WHERE {
   ?x   rdf:type              ?C  .
   ?C   rdf:type              owl:Restriction.
   ?C   owl:onProperty        :eats  .
   ?C   owl:someValuesFrom    :Plant
}
```
(Q2)

Moreover, the following queries Q3 and Q4 are legal for G since their BGPs can be mapped to the extended ontologies $\{x \sqsubseteq \mathsf{Mammal}\}$ and $\{x(grass)\}$, respectively, where variables are allowed in places of concepts and roles.

```
SELECT ?x WHERE { ?x rdfs:subClassOf :Mammal }
```
(Q3)

```
SELECT ?x WHERE { grass rdf:type ?x }
```
(Q4)

In contrast, the following query Q5 is not legal for G.

```
SELECT ?x WHERE {
   ?x  rdf:type  owl:TransitiveProperty  .
   ?x  rdf:type  owl:IrreflexiveProperty
}
```
(Q5)

According to the syntax restrictions of OWL 2 (see Section 3.1), an atomic role cannot be both transitive and irreflexive, and then $\{\mathsf{Tra}(x), \mathsf{Irr}(x)\}$ is not an extended OWL 2 ontology with variables. ♢

Semantics Given a legal RDF graph G, a *solution* for a legal query q under the entailment regime is a mapping σ from variables and blank nodes in q to nodes in G such that the application of σ to the BGP of q corresponds to an OWL 2 ontology that is entailed by the OWL 2 ontology corresponding to G.

Example 4.14. Consider the legal RDF graph G in Example 4.13, and the conjunctive query $q(x) = \exists y[\mathsf{eats}(x, y) \wedge \mathsf{Plant}(y)]$. Since $\mathcal{O} \models \mathsf{eats}(sheep, grass) \wedge \mathsf{Plant}(grass)$, we have that the mapping $\{x \mapsto sheep, y \mapsto grass\}$ is a solution for Q1 under the entailment regime. For any other mappings σ from $\{x, y\}$ to terms in G, $\mathcal{O} \not\models \mathsf{eats}(x, y)\sigma$. Therefore, *sheep* is the only answer to Q1 under the entailment regime. An easy observation is that Q1 captures the ground semantics of q. The solution for Q2 is $\{rabbit, sheep\}$ since $\mathcal{O} \models \exists \mathsf{eats}.\mathsf{Plant}(rabbit)$ and $\mathcal{O} \models \exists \mathsf{eats}.\mathsf{Plant}(grass)$. Thus, Q2 under the entailment regime captures the first-order semantics of q. ♢

Conjunctive Queries and SPARQL Conjunctive queries under the first-order semantics and SPARQL conjunctive queries under OWL 2 entailment regimes are incomparable query languages. On the one hand, some SPARQL conjunctive queries are not captured by conjunctive queries.

- SPARQL queries involving reserved vocabulary of RDF, RDFS or OWL other than rdf:type do not directly correspond to conjunctive queries, e.g. Q3.
- As mentioned before, some triple patterns do not directly correspond to first-order atoms. So SPARQL conjunctive queries involving such triple patterns do not directly correspond to conjunctive queries. For example, `SELECT` $?y$ `WHERE` { $?x$ rdf:type $?y$ } or `SELECT` $?y$ `WHERE` { $?x$ $?y$ $:grass$ }.

On the other hand, some conjunctive queries can not be captured by SPARQL under OWL 2 entailment regime. For example,

$$q(x) = \exists y_1 \exists y_2 \exists y_3 [A(x) \wedge R(x, y_1), \wedge S(y_1, y_2) \wedge S(y_2, y_3) \wedge S(y_3, y_1)].$$

This query contains a cycle y_1, y_2, y_3 of existentially quantified variables. As there were no OWL 2 ontologies that can present a cycle of existentially quantified variables, this query cannot be expressed in the entailment region. Note that the query can be directly translated into SPARQL atom by atom as the follows.

`SELECT` $?x$ `WHERE` {$?x$ rdf:type A`.` $?x$ `S` $?y_1$`.` $?y_1$ `S` $?y_2$`.` $?y_2$ `S` $?y_3$`.` $?y_3$ `S` $?y_1$`.`}

The semantics of the above query, however, is different from the conjunctive query due to the ground semantics of SPARQL.

Chapter 5

Query Answering in Ontologies

Conjunctive query answering over ontologies has received a great deal of attention in recent years. Its computational complexity has been thoroughly investigated for a wide range of KR languages and a number of practicable algorithms have been proposed in the literature and implemented in reasoning systems. In Section 5.1, we discuss relevant complexity results for conjunctive query in different KR languages. Next, in Section 5.2, we revisit existing techniques for conjunctive query answering over DL ontologies, such as tableau-based, materialisation-based, rewriting-based and combined approaches. Finally, we conclude this chapter by Section 5.3 discussing some prominent existing work on approximated reasoning in different KR languages.

5.1 Computational Complexity

The decision problem associated to conjunctive query answering is *conjunctive query entailment* (CQE), namely to decide whether $\mathcal{K} \models q(\mathbf{x})\sigma$ when given as input a conjunctive query q, a possible answer σ, and a knowledge base $\mathcal{K}$ expressed in a (fixed) language $\mathcal{L}$. This problem is well-known to be undecidable in general, even if q is restricted to be atomic and $\mathcal{L}$ is the language of existential rules (e.g., see (Dantsin et al., 2001)).

Decidability of CQE for knowledge bases stemming from OWL DL ontologies was established under the assumption that the query does not mention transitive relations (Rudolph and Glimm, 2010). Decidability of CQE for unrestricted OWL DL or OWL 2 ontologies remains an open problem. Even in the cases where CQE is decidable, it is typically of very high computational complexity. CQE is 2-ExpTime-complete for the expressive DLs $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$ (Glimm et al., 2008; Eiter et al., 2009). Hardness results for 2-ExpTime can be obtained already for relatively weak DL $\mathcal{ALCI}$ (Lutz, 2008), $\mathcal{SH}$ (Eiter et al., 2009), as well as for the logic Horn-$\mathcal{SROIQ}$

which underpins the Horn fragment of OWL 2 (Ortiz et al., 2011). CQE for $\mathcal{ALC}$ and $\mathcal{SHQ}$, where inverse roles are not involved, is ExpTime-complete (Lutz, 2008). Single exponential time results are also obtained for Horn DLs by disallowing complex role inclusion axioms: CQE is ExpTime-complete Horn-$\mathcal{SHOIQ}$, which underpins the Horn fragment of OWL DL (Ortiz et al., 2011).

Due to the high complexity of CQE, there has recently been an increasing interest in lightweight DLs for which CQE is computationally easier. Such lightweight DLs have been incorporated in the OWL 2 standard as *profiles* (Motik et al., 2009a). CQE in the OWL 2 EL profile is PSpace-complete (Stefanoni et al., 2014). Furthermore, the complexity of CQE drops to NP if complex role inclusions (with the exception only of transitivity and reflexivity) are disallowed in OWL 2 EL (Stefanoni and Motik, 2015). The latter complexity is rather benign since CQE over databases is already NP-hard. Finally, CQE for the OWL 2 QL profile is also NP-complete (Calvanese et al., 2007).

Regarding data complexity, CQE is coNP-complete for non-Horn DLs, such as $\mathcal{ALE}$ (Schaerf, 1993) and $\mathcal{SHIQ}$ (Ortiz et al., 2008; Glimm et al., 2008). In contrast, data complexity is PTime-complete for Horn DLs that can encode recursion, such as Horn-$\mathcal{SROIQ}$ and OWL 2 EL (Ortiz et al., 2011; Stefanoni et al., 2014). Finally, CQE in OWL 2 QL is known to be in LogSpace (and actually in AC^0) in data complexity as the problem is FO-rewritable (Calvanese et al., 2007).

The complexity of CQE is also well understood for rule-based knowledge representation languages. For plain datalog, it is ExpTime-complete in combined complexity and PTime-complete w.r.t. data complexity (Abiteboul et al., 1995). For disjunctive datalog, combined complexity increases to coNExpTime-complete, whereas data complexity increases to coNP-complete. We refer the reader to (Dantsin et al., 2001) for details. Datalog$^{\pm}$ refers to a family of decidable knowledge representation languages based on existential rules (Calì et al., 2012). This includes guarded (Calì et al., 2013), sticky (Calì et al., 2011), and acyclic (Cuenca Grau et al., 2013) datalog$^{\pm}$. The combined and data complexity in sticky TGDs is ExpTime-complete and in AC^0, respectively. The complexity increases to 2-ExpTime-complete in combined complexity and PTime-complete in data complexity for guarded TGDs. The extension of datalog$^{\pm}$ languages with disjunctive rules has been recently studied in (Alviano et al., 2012b; Bourhis et al., 2013).

Finally, we refer to ground query entailment (GCQE) as the problem of checking whether a substitution σ is a ground answer to $q(\mathbf{x}) = \exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$ w.r.t. $\mathcal{K}$. In ontology

languages that allow for existentially quantified rules or disjunctive rules, the restriction to ground answers typically makes CQE easier: the definition of ground answers means that GCQE can be trivially reduced to satisfiability checking. Consequently, GCQE is decidable for OWL 2.

5.2 Query Answering Approaches

Different query answering approaches have been implemented in existing systems. These systems are, however, limited in terms of expressiveness of the supported ontology languages, completeness of query answers and/or scalability of datasets they can efficiently process. We will revisit existing techniques for conjunctive query answering over DL ontologies. The tableau-based approaches in Section 5.2.1 are based on sound and complete tableau calculi for expressive DLs. Materialisation-based approaches described in Section 5.2.2 are widely used in practice for rule-based KR languages. Rewriting-based and combined approaches for lightweight Horn DLs are briefly reviewed in Section 5.2.3 and Section 5.2.4, respectively. Although, materialisation-based, rewriting-based, and combined approaches are highly scalable, their applications are limited as they are sound and complete for only a restricted fragment of OWL 2. If an input ontology is outside the supported ontology language, the computed query answers are not guaranteed to be complete. Finally, hybrid approaches that combine more than two kinds of approaches are discussed in Section 5.2.5. Hybrid approaches have been introduced for a trade off between scalability and expressivity of ontology languages.

5.2.1 Tableau-based Approaches

Tableau-based calculi have been successfully applied to traditional reasoning problems in DLs. Some off-the-shelf tableau-based DL reasoners, such as Pellet (Sirin et al., 2007), HermiT (Glimm et al., 2014) and Racer (Haarslev et al., 2012) also provide limited support for query answering. However, despite intensive efforts at optimisation, fully-fledged OWL reasoners can only deal with modestly-sized datasets. (Haarslev and Möller, 2008). Consequently, there has also been intensive work on optimising query answering in DL systems, including the filter-and-refine technique (Wandelt et al., 2010), ordering strategies of query atoms (Kollia and Glimm, 2013), and data summarisation (Dolby et al., 2009). Optimising query answering in DL reasoners is complementary to our approach, as the use of a more optimised DL reasoner could

significantly improve the performance of PAGOdA on hard queries that require fully-fledged reasoning.

Pellet Pellet (Sirin et al., 2007) was the first sound and complete OWL DL reasoner that supports conjunctive query answering. It implements a tableau-based algorithm with extensive optimisations. Pellet provides support for conjunctive queries under ground semantics. Under first-order semantics, Pellet is able to answer tree-shaped conjunctive queries using the rolling-up technique (see details in Section 10.2).

HermiT HermiT (Glimm et al., 2014) is a fully-fledged OWL 2 reasoner based on the hypertableau calculus (Motik et al., 2009c). It focuses on standard reasoning tasks, such as satisfiability, subsumption check and instance retrieval. In contrast to Pellet, it does not provide a SPARQL or conjunctive query answering API and it only supports conjunctive queries in the form of (complex) DL concepts. Moreover, it supports checking entailment of data assertions, which can be exploited to answer conjunctive queries.

Racer Racer was among the first reasoners to implement and optimise reasoning over ontologies with large datasets. It implements a tableau-based algorithm for the DL $\mathcal{SHIQ}$ and adopts a variety of optimisations for query answering. In particular, it implements the filter-and-refine technique for instance retrieval (Wandelt et al., 2010). It supports the query language *nRQL* (Haarslev et al., 2004), which provides support for conjunctive queries under the ground semantics extended with negation as failure and a number of aggregation operators.

5.2.2 Materialisation-based Approaches

RDF triple stores implement materialisation-based (a.k.a. forward chaining) reasoning algorithms, and queries are typically answered by evaluating them over the resulting materialisation. Materialisation-based approaches are applicable to typically fragments of datalog. Triple stores like Jena (McBride, 2001), Sesame (Broekstra et al., 2002) and Virtuoso (Erling and Mikhailov, 2009) aim at providing efficient query answering for large repositories over RDFS. Moreover, Oracle's native inference engine (Wu et al., 2008), OWLim (Bishop et al., 2011) and RDFox (Motik et al., 2014) provide extended suppport for ontologies in OWL 2 RL, where RDFox additionally supports arbitrary datalog programs over unary and binary predicates. Finally, DLV is a system for answer set programming implementing sound and complete reasoning

for (extensions of) disjunctive datalog (Leone et al., 2006). Although triple stores exhibit appealing scalability, they can support only restricted ontology languages. Similar to the case for fully-fledged OWL reasoners, improving the scalability of materialisation-based systems is complementary to our approach, and advances in this area can be directly exploited in our system PAGOdA.

Materialisation-based approaches are also known as forward chaining algorithm in datalog context. The state-of-the-art OWL 2 RL or datalog reasoner are typically materialisation-based. Backward chaining is an alternative approach to answer queries in datalog programs. Backward chaining with tabling is a well-known technique to evaluate queries in datalog programs (Tamaki and Sato, 1986; Sagonas and Swift, 1998). Magic set algorithm implements backward chaining in a forward-chaining fashion (Bancilhon et al., 1986). It is worth mentioning that magic set is related to the subset extraction technique in PAGOdA (see Section 9). The purpose for them are, however, different: magic set is used to make the materialisation process query-oriented so that it won't derive unrelated facts, whereas, our subset extraction starts with a fully materialised dataset and aims at identifying what facts are used in deriving relevant consequences.

5.2.3 Query Rewriting Approaches

Conjunctive query answering over Horn ontologies is often realised by means of *query rewriting*. Given an ontology $\mathcal{O}$, the rewriting of a conjunctive query q in $\mathcal{O}$ is a query q' that captures all information from $\mathcal{O}$ necessary to answer q over an arbitrary dataset. Such query q' is typically either a UCQ or a datalog query. Query rewriting allows to reuse optimised database management systems: UCQs can be evaluated using standard relational databases, whereas datalog queries correspondingly can be evaluated by datalog reasoners. In contrast to materialisation-based approaches, query rewriting techniques do not require modifying the data. Hence, query rewriting is well-suited for applications requiring frequent data updates. Although some of systems implemented the query rewriting approaches have been successfully applied in large scale applications, they are only applicable to deal with Horn ontologies. Furthermore, query rewriting sometimes suffers from an exponential blow-up in the size of the rewritten query (Calvanese et al., 2007).

Rewriting in *DL-Lite* Query rewriting approaches have been successfully applied to the *DL-Lite* family, for which rewritability into UCQ is guaranteed. *Perfect reformulation* is a way to rewrite a conjunctive query into UCQ w.r.t. OWL 2 QL

ontologies (Calvanese et al., 2007). Perfect reformulation has been implemented in QuOnto (Acciarri et al., 2005), which is further integrated in the OBDA system Mastro (Calvanese et al., 2011). A well-known issue of perfect reformulation is that the number of the rewritten queries may be exponential with respect to the size of the input query. Therefore, different optimisations have been proposed to reduce the size of the rewritten query, which are implemented in a number of existing *DL-Lite* systems like Owlgre (Stocker and Smith, 2008), Rapid (Chortaras et al., 2011), Prexto (Rosati, 2012), and Ontop (Bagosi et al., 2014).

Rewriting in $\mathcal{EL}$ Query rewriting into UCQs can be only applied to DLs with data complexity in AC^0. When data complexity increases to PTime, the rewritability into UCQ is not guaranteed. Conjunctive queries in $\mathcal{EL}$ can be rewritten into a datalog query (Rosati, 2007); concept queries in $\mathcal{EL}$ can be rewritten into non-recursive datalog rewriting if the query is FO-rewritable (Hansen et al., 2014).

A resolution-based query rewriting approach for $\mathcal{ELHIO}^\neg$ has been implemented in REQUIEM (Pérez-Urbina et al., 2010). The DL $\mathcal{ELHIO}^\neg$ covers both *DL-Lite* and $\mathcal{EL}$. Given an $\mathcal{ELHIO}^\neg$ ontology, a dataset and a conjunctive query, the rewriting is obtained by computing a saturated set of rules under certain resolution calculus, and then removing all rules containing functional terms. If the input ontology is expressed in *DL-Lite*$_\mathcal{R}$, the rewriting is a UCQ; if the input ontology is in *DL-Lite*$^+$, which is the resulting language of *DL-Lite*$_{core}$ by disallowing $C_\bot$ and additionally allowing $\exists R.A$ as concept, the rewriting is a linear datalog query; if the input ontology is in certain sub-languages of $\mathcal{ELHIO}$, the rewriting is a datalog query.

Rewriting in Expressive Horn-DLs Clipper implements a rewriting-based algorithm for query answering over Horn-$\mathcal{SHIQ}$ ontologies (Eiter et al., 2008). Different from other query-rewriting based systems listed in this section, Clipper also modifies the dataset during the query answering process. It takes as input a Horn-$\mathcal{SHIQ}$ ontology, a dataset and a conjunctive query q. First all transitivity axioms are eliminated by a known transformation (Kazakov, 2009). Then the ontology is saturated by a set of special tailored inferences and the dataset is materialised w.r.t. all datalog rules in the saturated ontology. Finally, the input query is rewritten against the existentially quantified rules in the ontology and the rewriting is evaluated against the augmented dataset. The saturated ontology and the query rewriting might be of exponential in the size of the input ontology and the input query.

5.2.4 Combined Approaches

A technique for conjunctive query answering over lightweight DLs that is receiving increasing attention is the so-called *combined approach*. Combined approaches have been applied to logics of the $\mathcal{EL}$ family, e.g. $\mathcal{ELH}^r_\bot$ — the extension of $\mathcal{ELH}$ with the bottom concept $\bot$ and range axioms (Lutz et al., 2009) and $\mathcal{ELHO}^r_\bot$(Stefanoni et al., 2013), as well as the *DL-Lite* family, like *DL-Lite*$^{\mathcal{N}}_{horn}$ — *DL-Lite*$_{horn}$ extended with number restrictions (Kontchakov et al., 2011) and *DL-Lite*$_{\mathcal{R}}$ (Lutz et al., 2013). Same as query rewriting approaches, the combined approaches are only applicable to Horn ontology languages.

In combined approaches the dataset is first augmented with new facts in a query-independent way to build (in polynomial time) a canonical model of the ontology. This model can be exploited for query answering in two equivalent ways. In the approach proposed by Lutz et al. (2009) or Kontchakov et al. (2011), an input query is first rewritten and then evaluated against the constructed model. Alternatively, in the work of Lutz et al. (2013) or Stefanoni et al. (2013), a query is first evaluated over the canonical model and then unsound answers are discarded by means of a polynomial time filtration process. One of the motivations for the second treatment is to avoid an exponential blow-up in query rewriting for certain combinations of DL constructors, e.g. *DL-Lite*$^{\mathcal{HN}}_{horn}$ (Kontchakov et al., 2011).

In this thesis, we exploit a filtration-based combined approach for $\mathcal{ELHO}^r_\bot$(Stefanoni et al., 2013) to compute the aggregated lower bound (see Section 7.3).

Canonical Model The basic idea to compute canonical model is rather similar for different combined approaches. Any input ontology can be translated into a knowledge base $\mathcal{K}$ as described in Section 3.1.2, and further to datalog program (with function symbols) $\mathsf{sk}(\mathcal{K})$ by Skolemisation (see Definition 2.2). The least Herbrand model of $\mathsf{sk}(\mathcal{K})$ is a universal model of $\mathcal{K}$, which is typically infinite. By replacing each functional terms $f^r_{ij}(\mathbf{t})$ in the least Herbrand model of $\mathsf{sk}(\mathcal{K})$ with a constant o^r_{ij}, we obtain a finite model of $\mathcal{K}$ that is logically stronger than the least Herbrand model. This finite model is the canonical model of $\mathcal{K}$, which is exactly the materialisation of a function-free datalog $D(\mathcal{K})$ program obtained from $\mathsf{sk}(\mathcal{K})$ by replacing functional terms $f^r_{ij}(\mathbf{x})$ with constants o^r_{ij}. All such constants are called *auxiliary* constants.

For some combined approaches, two auxiliary constants are introduced for each function symbol to obtain a polynomial procedure to filter out spurious answers, see e.g. filtration-based combined approach for *DL-Lite*$_{\mathcal{R}}$ (Lutz et al., 2013).

Filtration Stefanoni et al. (2013) has introduced a filtration-based combined approach for $\mathcal{ELHO}^r_\bot$. Several additional notations are needed to define the filtration procedure. We say that a constant a in $D(\mathcal{K})$ is *real-auxiliary* if no constant c exists in $\mathcal{K}$ such that $D(\mathcal{K}) \models a \approx c$. Since $D(\mathcal{K})$ is datalog, it takes only polynomial time in data complexity to determine if a constant is real-auxiliary. Let $\sim_q$ denote the smallest reflexive-transtive closure on the set of terms in q that is closed under the following relation.

$$(\mathsf{fork}) \qquad \frac{s' \sim_q t'}{s \sim_q t} \qquad \begin{array}{l} R(s, s') \text{ and } P(t, t') \text{ occur in } q, \text{ and} \\ \tau(s') \text{ is a real-auxiliary constant.} \end{array}$$

Clearly $\sim_q$ is an equivalent relation, which can be computed in polynomial time in the size of q. For any term t in q, let $[t]$ be the equivalent class of t w.r.t. $\sim_q$, and let γ be a mapping from each term t in q to an arbitrary but fixed representative of $[t]$.

Let $\mathcal{K}$, q be a knowledge base and a conjunctive query of the form $\exists \mathbf{x} \varphi(\mathbf{x}, \mathbf{y})$, respectively, and let π be an answer to $\varphi(\mathbf{x}, \mathbf{y})$ against the canonical model. The *auxiliary graph* of q and π is the directed graph $G = \langle V, E \rangle$ such that

- V contains a vertex $\gamma(t)$ for each term t in q such that $\pi(t)$ is a real-auxiliary constant; and
- E contains a directed edge $\langle \gamma(s), \gamma(t) \rangle$ for each atom of the form $R(s, t)$ in q such that $\{\gamma(s), \gamma(t)\} \subseteq V$.

The construction of the auxiliary graph can be done in polynomial time in the size of q. Now, we are ready to define the filtration process: π is a *spurious* answer iff any of the following conditions holds. Clearly, the procedure for a possible answer π is in polynomial time in the size of q.

(a) Free variable $x \in \mathbf{x}$ exists such that $\pi(x)$ is an auxiliary constant.

(b) Terms s and t occurring in q exist such that $s \sim_q t$ but $D(\mathcal{K}) \not\models \pi(s) \approx \pi(t)$.

(c) The auxiliary graph of q and π contains a cycle.

5.2.5 Hybrid Approaches

PAGOdA is based on a hybrid approach that combines a datalog reasoner and a fully-fledged OWL 2 reasoner. Hydrowl is very similar to PAGOdA in spirit in that it combines an OWL 2 RL reasoner with a query rewriting system, and a fully-fledged DL reasoner to answer conjunctive queries over OWL 2 knowledge bases

(Stoilos, 2014b). The techniques in Hydrowl are, however, rather different to those in PAGOdA. Hydrowl uses two different query answering strategies. The first one is based on repairing (Stoilos, 2014a) and query rewriting, which is applicable only to ontologies for which a suitable repair exists. The second strategy exploits a set of atomic queries, called *query base* that can be fully answered using the triple store for a given ontology and an arbitrary dataset. The query base is computed in a pre-processing phase before answering any queries. Given an input query q, Hydrowl checks if q is "covered" by query base (Stoilos and Stamou, 2014). If it is, then q can be completely evaluated using the OWL 2 RL reasoner; otherwise, the fully-fledged reasoner is used to answer q. The computation of the query base, however, does not appear to be correct in general,[1] and we believe that this accounts for the incompleteness of Hydrowl in some of our tests (see Section 12.3.1). In contrast to PAGOdA, Hydrowl doesn't improve the performance of satisfiability check, in the sense that it fully relies on HermiT to check satisfiability. However, satisfiability check over large datasets is often infeasible in practice.

5.3 Approximated Reasoning

The problem to approximate expressive languages into less expressive ones are widely explored and well-understood. *Theory approximation* was first described in the seminal paper by (Selman and Kautz, 1996). The idea in theory approximation is to approximate a logical theory Σ by two theories Σ_{lb} (the *model lower bound*) and Σ_{ub} (the *model upper bound*) such that $\Sigma_{lb} \models \Sigma \models \Sigma_{ub}$, both Σ_{lb} and Σ_{ub} are in a "more tractable" language than Σ, and Σ_{lb} and Σ_{ub} are "as close as possible" to Σ. Selman and Kautz (1996) studied this problem for Σ in propositional logic and the bounds expressed in its Horn fragment. Val (2005) studied the problem for first-order logic. This line of research has focused mostly on the computation of the "best" model upper bounds; however, in terms of query answers, the model lower bounds can be exploited to compute query upper bounds, which have received little attention. The idea of transforming the ontology, data, and/or query to obtain upper bound query answers has also received some attention in the semantic web community. Efficient approximation strategies for rule-based KR languages are again complementary to our approach, as they can be exploited by PAGOdA in order to refine lower and upper bound query answers.

[1] Stoilos (2014b) mentions "a limitation in automatically extracting [the atomic queries]".

Screech The SCREECH system (Tserendorj et al., 2008) first exploits the KAON2 reasoner (Hustadt et al., 2007) to rewrite a $\mathcal{SHIQ}$ ontology into a (possibly exponential size) disjunctive datalog program in such a way that ground answers to queries are preserved. It provides no guarantee for certain answers of conjunctive queries. Subsequently, Screech exploits different (unsound or incomplete) techniques to transform disjunctive datalog into plain datalog. In this way, Screech computes only an approximation of the set of answers. The exponential blow-up in the translation to disjunctive datalog means that, in practice, KAON2 may be unable to process large or complex ontologies; for example, KAON2 was reported to fail on the DOLCE ontology (Motik and Sattler, 2006).

SHER Another system SHER implements a tableau-based algorithm complete for $\mathcal{SHIN}$ that provides also effectively instance retrieval. SHER system uses summarisation (see Section 10.1.1 for details) to efficiently compute an upper bound answer, with exact answers then being computed via successive relaxations (Dolby et al., 2009). The technique has been shown to be scalable in practice, but it is only known to be applicable to $\mathcal{SHIN}$ and instance queries.

TrOWL TrOWL (Thomas et al., 2010) exploits approximation techniques to transform an OWL 2 ontology into OWL 2 QL (Pan et al., 2009). The approximation is based on two concepts—axiom set and entailment set. The *axiom set* for a given ontology $\mathcal{O}$ in *DL-Lite* is the set of *DL-Lite* axioms that can be constructed in the signature of $\mathcal{O}$, where the *entailment set* is a subset of the axiom set that are entailed by $\mathcal{O}$. As a result, the entailment set is an under-approximation of $\mathcal{O}$. Given an input query q, TrOWL adds query-dependent axioms to the entailment set to obtain a so-called *enriched entailment set*. The set of query answers to q w.r.t. the enriched entailment set is an upper bound of answers to q w.r.t. the input ontology. Each entailment test requires the use of a fully-fledged OWL reasoner, which can be expensive; also, the enriched entailment set cannot be reused as it is both query-dependent and dataset-dependent.

An alternative way to approximate an arbitrary OWL 2 ontology $\mathcal{O}$ into an OWL 2 QL ontology $\mathcal{O}'$ has also been proposed by Wandelt et al. (2010) to provide an upper bound of query answers. Each axiom $C \sqsubseteq D$ in $\mathcal{O}$ is transformed into an OWL 2 QL axiom $C' \sqsubseteq D'$, where C is subsumed by C' and D' is subsumed by D (w.r.t. $\mathcal{O}$). The transformation algorithm, however, is non-deterministic and there can be exponentially many C' and D' satisfying the required properties. Furthermore, as

reported in this work, it is often the case that for a given dataset $\mathcal{D}$ such that $\mathcal{O} \cup \mathcal{D}$ is satisfiable, $\mathcal{O}' \cup \mathcal{D}$ is unsatisfiable, regardless of the choices made when computing $\mathcal{O}'$. The large degree of non-determinism means that computing $\mathcal{O}'$ can be expensive, even for small ontologies.

So far, we have discussed about program approximation. On the other hand, there have been a lot of existing works on query approximation. Techniques to provide lower and upper bound answers for SPARQL queries have been proposed for OWL ontologies (Glimm et al., 2015), as well as an optimisation called query extension. Query approximation has also been extensively explored in databases. Some prior works are quite similar to PAGOdA in the spirit of computing a lower and an upper bound of query answers, such as, Barceló et al. (2012)'s work in the relational setting, Fink et al. (2013)'s work in the probabilistic and relational setting, and Gatterbauer and Suciu (2015)'s work on query approximation by data and query manipulation.

Part II

The "Pay-as-you-go" Approach

Chapter 6

Overview

In this chapter we provide a high-level overview of our approach to query answering. We assume the availability of two reasoners:

- a *datalog reasoner* that is sound and complete for answering conjunctive queries over datalog knowledge bases; and
- a *fully-fledged reasoner* that is sound and complete for answering a given class of conjunctive queries $\mathcal{Q}$ (which includes the unsatisfiability query) w.r.t. knowledge bases in a given ontology language $\mathcal{L}$.

We will describe our approach in its most general form, where we make no assumptions about the two reasoners, treating them as "black-box" query answering procedures.

The kind of queries and knowledge bases that can be dealt with using this approach ultimately depends on the capabilities of the fully-fledged reasoner. For instance, OWL 2 reasoners can typically process arbitrary OWL 2 knowledge bases; however, the query language is limited to tree-shaped queries, e.g. (Sirin et al., 2007). In turn, the scalability of our approach ultimately depends on how much of the reasoning workload can be delegated to the datalog reasoner; our goal is to delegate the bulk of the computation to the datalog reasoner and to restrict the (expensive) use of the fully-fledged reasoner to the bare minimum.

Here, and in the rest of Part II, we fix an arbitrary normalised knowledge base $\mathcal{K} = \Sigma_{\mathcal{K}} \cup \mathcal{D}_{\mathcal{K}}$. Given an arbitrary conjunctive query q (which may be the special unsatisfiability query) containing only symbols from $\mathcal{K}$, the core of our approach relies on exploiting the datalog reasoner for accomplishing the following tasks:

- **lower and upper bound computation**, where we exploit the datalog reasoner to compute both a lower bound L^q and an upper bound U^q of the certain answers to q w.r.t. $\mathcal{K}$. If these bounds match (i.e. $L^q = U^q$), then the

query has been fully answered by the datalog reasoner; otherwise, the difference $G^q = U^q \setminus L^q$ provides a set of "gap" answers that need to be verified using the fully-fledged reasoner. The relevant techniques for computing these bounds are described in Chapter 7 and 8; and

- **knowledge base subset computation**, where we exploit the datalog reasoner to compute a (hopefully small) subset $\mathcal{K}^q$ of $\mathcal{K}$ that is sufficient to check if answers in G^q are in $\mathsf{cert}(q, \mathcal{K})$; that is, $\sigma \in \mathsf{cert}(q, \mathcal{K})$ iff $\sigma \in \mathsf{cert}(q, \mathcal{K}^q)$ for each $\sigma \in G^q$. The details on how to compute such $\mathcal{K}^q$ are given in Chapter 9.

We then proceed according to the following steps when given an input query q:

Step 1. Check satisfiability of $\mathcal{K}$.

(a) Compute bounds $L^\bot$ and $U^\bot$ for the unsatisfiability query $\bot$. If $L^\bot \neq \emptyset$, then terminate and report that $\mathcal{K}$ is unsatisfiable. If $U^\bot = \emptyset$, then proceed to Step 2 ($\mathcal{K}$ is satisfiable).

(b) Compute the subset $\mathcal{K}^\bot$ of $\mathcal{K}$.

(c) Use the fully-fledged reasoner to check the satisfiability of $\mathcal{K}^\bot$. To minimise the computational workload of the fully-fledged reasoner, we proceed as follows:

i. Construct a summary of $\mathcal{K}^\bot$ (see Section 10.1.1), and use the fully-fledged reasoner to check if it is satisfiable; if it is, proceed to Step 2 ($\mathcal{K}$ is satisfiable).

ii. Use the fully-fledged reasoner to check the satisfiability of $\mathcal{K}^\bot$; if it is unsatisfiable, then terminate and report that $\mathcal{K}$ is unsatisfiable. Otherwise, proceed to Step 2 ($\mathcal{K}$ is satisfiable).

Step 2. Compute bounds L^q and U^q. If $G^q = \emptyset$, then terminate and return L^q. Otherwise, proceed to Step 3.

Step 3. Compute the subset $\mathcal{K}^q$ of $\mathcal{K}$ based on $\mathcal{K}$, q, and G^q.

Step 4. For each $\sigma \in G^q$, use the fully-fledged reasoner to check whether $\mathcal{K}^q \models q\sigma$. To minimise the computational workload, this step is carried out as follows:

(a) Construct a summary of $\mathcal{K}^q$ (see Section 10.1.1) by merging constants. For each $\sigma \in G^q$, use the fully-fledged reasoner to check whether the summary of σ is not a certain answer to q w.r.t. the summary of $\mathcal{K}^q$, and remove σ from G^q if it is not the case.

$$\mathsf{Mammal}(tiger) \quad \text{(D1)} \qquad \mathsf{Mammal}(wolf) \quad \text{(D6)} \qquad \mathsf{Mammal}(howler) \quad \text{(D11)}$$
$$\mathsf{Mammal}(lion) \quad \text{(D2)} \qquad \mathsf{MeatEater}(wolf) \quad \text{(D7)} \qquad \mathsf{Folivore}(howler) \quad \text{(D12)}$$
$$\mathsf{MeatEater}(python) \quad \text{(D3)} \qquad \mathsf{eats}(wolf, sheep) \quad \text{(D8)} \qquad \mathsf{Mammal}(a\_hare) \quad \text{(D13)}$$
$$\mathsf{eats}(python, rabbit) \quad \text{(D4)} \qquad \mathsf{Herbivore}(sheep) \quad \text{(D9)} \qquad \mathsf{Folivore}(a\_hare) \quad \text{(D14)}$$
$$\mathsf{Herbivore}(rabbit) \quad \text{(D5)} \qquad \mathsf{eats}(sheep, grass) \quad \text{(D10)} \qquad \mathsf{eats}(a\_hare, willow) \quad \text{(D15)}$$

$$\mathsf{Carnivore}(x) \rightarrow \mathsf{Mammal}(x) \quad \text{(R1)}$$
$$\mathsf{Herbivore}(x) \rightarrow \mathsf{Mammal}(x) \quad \text{(R2)}$$
$$\mathsf{Folivore}(x) \wedge \mathsf{MeatEater}(x) \rightarrow \bot \quad \text{(R3)}$$
$$\mathsf{Herbivore}(x) \wedge \mathsf{eats}(x, y) \rightarrow \mathsf{Plant}(y) \quad \text{(R4)}$$
$$\mathsf{Mammal}(x) \rightarrow \mathsf{Herbivore}(x) \vee \mathsf{MeatEater}(x) \quad \text{(R5)}$$
$$\mathsf{MeatEater}(x) \rightarrow \exists y[\mathsf{eats}(x, y) \wedge \mathsf{Herbivore}(y)] \quad \text{(R6)}$$
$$\mathsf{Mammal}(x) \rightarrow \exists y\, \mathsf{eats}(x, y) \quad \text{(R7)}$$
$$\mathsf{Folivore}(x) \rightarrow \exists y[\mathsf{eats}(x, y) \wedge \mathsf{Leaf}(y)] \quad \text{(R8)}$$
$$\mathsf{Leaf}(x) \rightarrow \mathsf{Plant}(x) \quad \text{(R9)}$$

$$q_{ex}(x) = \exists y[\mathsf{eats}(x, y) \wedge \mathsf{Plant}(y)]$$
$$\mathsf{cert}(q_{ex}, \mathcal{K}_{ex}) = \{rabbit, sheep, howler, a\_hare\}$$

Figure 6.1: Running example knowledge base $\mathcal{K}_{ex}$ and query q_{ex}

(b) Compute a dependency relation between the remaining answers in G^q such that if a possible answer σ depends on a possible answer π, and σ is a spurious answer, then so is π. (See Section 10.1.2).

(c) Remove any remaining spurious answers from G^q, where an answer is spurious if it is not entailed by $\mathcal{K}^q$ or if a spurious answer depends on it; use the fully-fledged reasoner to check relevant entailments, arranging checks by heuristics w.r.t. the dependency relation.

Step 5. Return $L^q \cup G^q$.

In the following chapters, we describe each of these steps formally. We will also introduce a number of improvements and optimisations, some of which rely on the additional assumption that the datalog reasoner is materialisation-based—that is, for a datalog knowledge base $\mathcal{H}$ and query q, it computes query answers $\mathsf{cert}(q, \mathcal{H})$ by first computing the materialisation $\mathsf{skChase}(\mathcal{H})$ as defined in Definition 2.3 and then evaluating q over the resulting materialisation. This is a reasonable assumption in practice since most datalog reasoners in semantic web applications (e.g., OWLim,

RDFox, Oracle's native inference engine) are materialisation-based. In such cases, we further assume that we have direct access to the materialisation.

We will illustrate all our techniques using a running example consisting of the knowledge base $\mathcal{K}_{ex} = \Sigma_{\mathcal{K}_{ex}} \cup \mathcal{D}_{\mathcal{K}_{ex}}$ and the query q_{ex} given in Figure 6.1. Note that rules (R6) and (R8) in $\Sigma_{\mathcal{K}_{ex}}$ are not normalised; however, they can be easily brought into normal form by introducing fresh binary predicates eats_H and eats_L as described in Section 3.1.2:

$$\mathsf{MeatEater}(x) \rightarrow \exists y\, \mathsf{eats}_H(x, y) \quad \text{(R6a)}$$
$$\mathsf{eats}(x, y) \wedge \mathsf{Herbivore}(y) \rightarrow \mathsf{eats}_H(x, y) \quad \text{(R6b)}$$
$$\mathsf{eats}_H(x, y) \rightarrow \mathsf{eats}(x, y) \quad \text{(R6c)}$$
$$\mathsf{eats}_H(x, y) \rightarrow \mathsf{Herbivore}(y) \quad \text{(R6d)}$$
$$\mathsf{Folivore}(x) \rightarrow \exists y\, \mathsf{eats}_L(x, y) \quad \text{(R8a)}$$
$$\mathsf{eats}(x, y) \wedge \mathsf{Leaf}(y) \rightarrow \mathsf{eats}_L(x, y) \quad \text{(R8b)}$$
$$\mathsf{eats}_L(x, y) \rightarrow \mathsf{eats}(x, y) \quad \text{(R8c)}$$
$$\mathsf{eats}_L(x, y) \rightarrow \mathsf{Leaf}(y) \quad \text{(R8d)}$$

Our core techniques described in Chapter 7-9 are applicable to any knowledge base and any query. In order to simplify the presentation of our definitions and technical results of those sections we fix, in addition to the knowledge base $\mathcal{K} = \Sigma_{\mathcal{K}} \cup \mathcal{D}_{\mathcal{K}}$, an arbitrary query $q(\mathbf{x}) = \exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$ (which may be the unsatisfiability query $\bot$).

Chapter 7

Lower Bound Computation

A straightforward way to compute lower bound answers using the datalog reasoner is to evaluate q w.r.t. the datalog subset of $\mathcal{K}$ consisting of all facts in $\mathcal{D}_\mathcal{K}$ and datalog rules in $\Sigma_\mathcal{K}$. By the monotonicity of first-order logic (Besnard, 1989) all certain answers w.r.t. such subset are also certain answers w.r.t. $\mathcal{K}$. Furthermore, if the subset is unsatisfiable, then so is $\mathcal{K}$.

Example 7.1. The datalog subset of our example $\mathcal{K}_{ex}$ consists of rules (R1)–(R4), (R9), (R6b)–(R6d), and (R8b)–(R8d) together with all facts (D1)–(D15). The materialisation of the datalog subset of $\mathcal{K}_{ex}$ results in the following dataset:[1]

$$\mathcal{D}_{ex} \cup \{\mathsf{Mammal}(rabbit), \mathsf{Mammal}(sheep), \mathsf{Plant}(grass)\}.$$

When evaluating q_{ex} against the materialisation we obtain *sheep* as an answer. ♢

This basic lower bound can be rather imprecise in practice since rules featuring disjunction or existential quantification typically abound in OWL 2 ontologies. To improve this bound, we exploit techniques that allow us to deterministically derive (also via datalog reasoning) additional consequences from $\mathcal{K}$ that do not follow from its datalog subset.

7.1 Program Shifting

To deal with disjunctive rules, we adopt a variant of *shifting*—a polynomial program transformation commonly used in Answer Set Programming (Eiter et al., 2004). We next illustrate the intuition behind this transformation with an example.

[1]Through out the running example, we ignore derived facts over predicates introduced during the normalisation for brevity.

Example 7.2. Let us consider the information in $\mathcal{K}_{ex}$ about an Arctic hare *a_hare*. From (R3) and (D14), one can deduce that *a_hare* is not a $\mathsf{MeatEater}$, and it further follows by rule (R5) and fact (D13) that *a_hare* is a $\mathsf{Herbivore}$. Since *a_hare* eats *willow* in (D15), we can deduce $\mathsf{Plant}(willow)$ from (R4) and hence *a_hare* is an answer to q_{ex}. Although (R5) is a disjunctive rule, this reasoning process is fully deterministic and can be captured in datalog. To this end, we introduce a predicate $\overline{\mathsf{MeatEater}}$ which intuitively stands for the complement of $\mathsf{MeatEater}$. We can then extend the datalog subset of $\mathcal{K}_{ex}$ with rules encoding the intended meaning of the fresh predicate. In particular, $(\overline{R3})$ and $(\overline{R5})$ are two such rules, which are obtained from (R3) and (R5), respectively.

$$\mathsf{Folivore}(x) \rightarrow \overline{\mathsf{MeatEater}}(x) \qquad (\overline{R3})$$

$$\mathsf{Mammal}(x) \wedge \overline{\mathsf{MeatEater}}(x) \rightarrow \mathsf{Herbivore}(x) \qquad (\overline{R5})$$

We can exploit these rules to derive $\overline{\mathsf{MeatEater}}(a\_hare)$ and $\mathsf{Herbivore}(a\_hare)$. Furthermore, we can derive $\mathsf{Plant}(willow)$ by (R4). ♢

We now define the shifting transformation formally.

Definition 7.3. Let r be a normalised disjunctive datalog rule of the form $\beta_1 \wedge \ldots \wedge \beta_n \rightarrow \gamma_1 \vee \ldots \vee \gamma_m$. For each predicate P in r let $\overline{P}$ be a fresh predicate of the same arity. Furthermore, given an atom $\alpha = P(\mathbf{t})$, let $\overline{\alpha}$ be the atom $\overline{P}(\mathbf{t})$. The shifting of r, written $\mathsf{shift}(r)$, is the following set of rules:

- if r of the form ($\bot$-`Norm`), then

$$\mathsf{shift}(r) = \{r\} \cup \{\beta_1 \wedge \cdots \wedge \beta_{i-1} \wedge \beta_{i+1} \wedge \cdots \wedge \beta_n \rightarrow \overline{\beta}_i \mid 1 \leq i \leq n\};$$

- if r of the form ($\vee$-`Norm`), then $\mathsf{shift}(r)$ consists of the following rules: *(i)* the rule (S1); *(ii)* all rules (S2) for each $1 \leq j \leq m$; and *(iii)* all rules (S3) for each $1 \leq i \leq n$ such that each variable in β_i also occurs in some other atom in the rule.

$$\beta_1 \wedge \cdots \wedge \beta_n \wedge \overline{\gamma}_1 \wedge \cdots \wedge \overline{\gamma}_m \rightarrow \bot \qquad \text{(S1)}$$

$$\beta_1 \wedge \cdots \wedge \beta_n \wedge \overline{\gamma}_1 \wedge \cdots \wedge \overline{\gamma}_{j-1} \wedge \overline{\gamma}_{j+1} \wedge \cdots \wedge \overline{\gamma}_m \rightarrow \gamma_j \qquad \text{(S2)}$$

$$\beta_1 \wedge \cdots \wedge \beta_{i-1} \wedge \beta_{i+1} \wedge \cdots \wedge \beta_n \wedge \overline{\gamma}_1 \wedge \cdots \wedge \overline{\gamma}_m \rightarrow \overline{\beta}_i \qquad \text{(S3)}$$

Let Σ be a set of normalised disjunctive datalog rules. Then, the shifting of Σ is defined as the following set of datalog rules:

$$\mathsf{shift}(\Sigma) = \bigcup_{r \in \Sigma} \mathsf{shift}(r) \qquad \diamondsuit$$

Note that shifting is a polynomial transformation. For a disjunctive datalog rule r with n atoms in the body and m atoms in the head, $\mathsf{shift}(r)$ contains at most $m+n+1$ datalog rules. Furthermore, as shown in the following theorem, it is also sound.

Theorem 7.4. *Let $\Sigma_{\mathcal{K}}^{DD}$ be the subset of disjunctive datalog rules in $\Sigma_{\mathcal{K}}$; furthermore, let $\mathcal{K}' = \mathsf{shift}(\Sigma_{\mathcal{K}}^{DD}) \cup \mathcal{D}_{\mathcal{K}}$. Then, $\mathsf{cert}(q, \mathcal{K}') \subseteq \mathsf{cert}(q, \mathcal{K})$.*

Proof. Let $\{\mathcal{B}^i\}$ be the Skolem chase sequence of $\mathcal{K}'$ defined in Definition 2.3, and let L be the natural number such that $\mathsf{skChase}(\mathcal{K}') = \mathcal{B}^L$ (recall that $\mathcal{K}'$ is a datalog knowledge base and hence its Skolem chase is finite). We show by induction that the following properties hold for each $0 \leq i \leq L$ and each $\alpha \in \mathcal{B}^i$:

(a) if $\alpha = \bot$, then $\mathcal{K}$ is unstisfiable;

(b) if $\alpha = P(\mathbf{a})$, then $\mathcal{K} \models P(\mathbf{a})$; and

(c) if $\alpha = \overline{P}(\mathbf{a})$, then $\mathcal{K} \models \neg P(\mathbf{a})$.

Base case: Clearly, $\mathcal{B}^0 = \mathcal{D}_{\mathcal{K}}$ and the properties trivially follow from the fact that $\mathcal{D}_{\mathcal{K}} \subseteq \mathcal{K}$.

Inductive step: Assume that properties (a)–(c) hold for every $\alpha \in \mathcal{B}^i$. We show that they also hold for every $\alpha \in \mathcal{B}^{i+1} \setminus \mathcal{B}^i$. There must exist a rule $r' \in \mathcal{K}'$ and a substitution σ such that $\mathcal{B}^i \models \mathsf{body}(r')\sigma$ and $\alpha = \mathsf{head}(r')\sigma$. Since every atom in $\mathsf{body}(r')\sigma$ is in $\mathcal{B}^i$, then properties (a)-(c) hold for all these atoms by the induction hypothesis. Furthermore, there must exist a rule $r \in \mathcal{K}$ of the form of $\beta_1 \wedge \cdots \wedge \beta_n \rightarrow \gamma_1 \vee \cdots \vee \gamma_m$ such that $r' \in \mathsf{shift}(r)$.

(a) If $\alpha = \bot$, we distinguish two cases. *(i)* $\mathsf{head}(r) = \bot$, in which case $r = r'$ and by the induction hypothesis, $\mathcal{K} \models \{\beta_1\sigma, \ldots, \beta_n\sigma\}$ and hence $\mathcal{K} \models \bot$; *(ii)* $\mathsf{head}(r) \neq \bot$, in which case r' is of the form (S1) and $\beta_1\sigma, \ldots, \beta_n\sigma$ and $\overline{\gamma}_1\sigma, \ldots, \overline{\gamma}_m\sigma$ are in $\mathcal{B}^i$. By the induction hypothesis, $\mathcal{K}$ entails $\beta_1\sigma, \ldots, \beta_n\sigma$ and $\neg\gamma_1\sigma, \ldots, \neg\gamma_m\sigma$, and hence $\mathcal{K} \models \neg r\sigma$. Since $r \in \mathcal{K}$, we obtain that $\mathcal{K}$ is unsatisfiable.

(b) If $\alpha = P(\mathbf{a})$, then r' is of the form (S2) and $\gamma_j\sigma = P(\mathbf{a})$. Hence, $\mathcal{B}^i$ contains all atoms $\beta_1\sigma, \ldots, \beta_n\sigma$, $\overline{\gamma}_1\sigma, \ldots \overline{\gamma}_{j-1}\sigma$ and $\overline{\gamma}_{j+1}\sigma, \ldots, \overline{\gamma}_m\sigma$. By induction hypothesis, $\mathcal{K}$ entails $\beta_1\sigma, \ldots, \beta_n\sigma$, $\neg\gamma_1\sigma, \ldots, \neg\gamma_{j-1}\sigma$, and $\neg\gamma_{j+1}\sigma, \ldots, \neg\gamma_m\sigma$. Since $r \in \mathcal{K}$ and $\gamma_j\sigma = P(\mathbf{a})$ it must be the case that $\mathcal{K} \models P(\mathbf{a})$.

(c) If $\alpha = \overline{P}(\mathbf{a})$, we have the following cases. *(i)* $\mathsf{head}(r) = \bot$, in which case by induction $\mathcal{K} \models \{\beta_1\sigma, \ldots, \beta_{j-1}\sigma, \beta_{j+1}\sigma, \ldots, \beta_n\sigma\}$; but then, since $\beta_1 \wedge \cdots \wedge \beta_n \rightarrow \bot$ is also a rule in $\mathcal{K}$, we obtain that $\mathcal{K} \models \neg\beta_j\sigma$, as required. *(ii)* $\mathsf{head}(r) \neq \bot$, in which case r' is of the form (S3) and $\beta_j\sigma = P(\mathbf{a})$; then, $\mathcal{B}^i$ contains all atoms $\beta_1\sigma, \ldots, \beta_{j-1}\sigma$, $\beta_{j+1}\sigma, \ldots, \beta_n\sigma$, and $\overline{\gamma}_1\sigma, \ldots, \overline{\gamma}_m\sigma$ and by the induction hypothesis $\mathcal{K}$ entails atoms $\beta_1\sigma, \ldots, \beta_{j-1}\sigma$, $\beta_{j+1}\sigma, \ldots, \beta_n\sigma$ and $\neg\gamma_1\sigma, \ldots, \neg\gamma_m\sigma$. Since $r \in \mathcal{K}$, we obtain that $\mathcal{K} \models \neg P(\mathbf{a})$.

If $q = \bot$, then the theorem follows from property (a). Otherwise, let $q(\mathbf{x}) = \exists\mathbf{y}\, \varphi(\mathbf{x}, \mathbf{y})$ and let σ be a certain answer to q w.r.t. $\mathcal{K}'$. Since $\mathcal{K}'$ is datalog, there exists a substitution σ' from $\mathbf{y}$ to constants in $\mathcal{K}'$ such that $\mathcal{B}^L \models \varphi(\mathbf{x}, \mathbf{y})\sigma\sigma'$. Then, by (b) we have $\mathcal{K} \models \alpha$ for each α in $\varphi(\mathbf{x}, \mathbf{y})\sigma\sigma'$, and hence $\mathcal{K} \models q(\mathbf{x})\sigma$. □

Note that shifting only captures some of the consequences related to the disjunctive datalog rules in $\mathcal{K}$ and hence it is an incomplete program transformation.

Example 7.5. Consider a disjunctive datalog knowledge base consisting of the fact $\mathsf{GreenSeaTurtle}(\mathit{turtle})$, the rules (R1), (R2) and

$$\mathsf{GreenSeaTurtle}(x) \rightarrow \mathsf{Herbivore}(x) \vee \mathsf{Carnivore}(x).$$

Clearly, $\mathsf{Mammal}(\mathit{turtle})$ follows from the knowledge base. The shifting consists of fact $\mathsf{GreenSeaTurtle}(\mathit{turtle})$ and the following rules, where predicates Carnivore, GreenSeaTurtle, Herbivore and Mammal have been abbreviated as, respectively, C, G, H and M:

$$\begin{array}{lll}
\mathsf{C}(x) \wedge \overline{\mathsf{M}}(x) \rightarrow \bot & \mathsf{C}(x) \rightarrow \mathsf{M}(x) & \overline{\mathsf{M}}(x) \rightarrow \overline{\mathsf{C}}(x) \\
\mathsf{H}(x) \wedge \overline{\mathsf{M}}(x) \rightarrow \bot & \mathsf{H}(x) \rightarrow \mathsf{M}(x) & \overline{\mathsf{M}}(x) \rightarrow \overline{\mathsf{H}}(x) \\
\mathsf{G}(x) \wedge \overline{\mathsf{H}}(x) \wedge \overline{\mathsf{C}}(x) \rightarrow \bot & \mathsf{G}(x) \wedge \overline{\mathsf{H}}(x) \rightarrow \mathsf{C}(x) & \mathsf{G}(x) \wedge \overline{\mathsf{C}}(x) \rightarrow \mathsf{H}(x) \\
\overline{\mathsf{H}}(x) \wedge \overline{\mathsf{C}}(x) \rightarrow \overline{\mathsf{G}}(x) & &
\end{array}$$

It can be checked that fact $\mathsf{Mammal}(\mathit{turtle})$ does not follow from the shifting. ♢

7.2 The Combined Approach for $\mathcal{ELHO}^r_\bot$

Existentially quantified rules are ubiquitous in large-scale and complex ontologies, especially in life sciences applications. The EL profile of OWL 2 was specifically designed for such applications, and many large ontologies used in practice can be

seen as consisting of a large EL backbone extended with a small number of axioms outside the profile.

Given the prevalence of EL axioms in realistic ontologies, it is natural to consider the OWL 2 EL subset of $\mathcal{K}$ for computing lower bound answers. CQ answering for OWL 2 EL is, however, PSPACE-complete (Stefanoni et al., 2014) and no system currently supports CQ answering for the whole of OWL 2 EL. Complexity, however, drops to NP in the case of $\mathcal{ELHO}^r_\bot$ (Stefanoni et al., 2014). In our setting, the restriction to $\mathcal{ELHO}^r_\bot$ ontologies has the added practical benefit that we can exploit the so-called *combined approach* to delegate most of the computational work associated with CQ answering to a datalog reasoner (Stefanoni et al., 2013; Lutz et al., 2009)—a technique currently supported by systems such as KARMA.[2] Although datalog-based CQ answering techniques are also available for richer languages, such as the extension of $\mathcal{ELHO}^r_\bot$ with inverse roles, the resulting datalog programs can be hard to compute and are of exponential size in the worst case (Pérez-Urbina et al., 2010). This is in contrast to the combined approach to $\mathcal{ELHO}^r_\bot$, where the relevant datalog programs can be straightforwardly constructed without the need for reasoning, and are of linear size (see Related Work section for further details).

To compute query answers that depend on existentially quantified rules, we consider the subset of $\mathcal{ELHO}^r_\bot$ rules in $\mathcal{K}$. As mentioned in Section 5.2.4, the combined approach for $\mathcal{ELHO}^r_\bot$ can be conceptualised as a three-step process.

1. The first step is to compute the canonical model M of the $\mathcal{ELHO}^r_\bot$ knowledge base. If M contains $\bot$, then the knowledge base is unsatisfiable. Otherwise M is a model of the knowledge base.

2. The second step is to evaluate the query q over M. This model M, however, is not universal. Thus, the evaluation of queries over M may lead to unsound answers w.r.t. the $\mathcal{ELHO}^r_\bot$ knowledge base.

3. In the third step, unsound answers obtained from the second step are discarded using a polynomial time *filtration algorithm*.

We next specify the transformation from $\mathcal{ELHO}^r_\bot$ knowledge bases to datalog used in the first step, which will also be exploited later on in Chapter 8 for computing upper bound query answers. The filtration process has been briefly described in Section 5.2.4. Here, we assume the availability of a procedure soundAnswers that

[2] http://www.cs.ox.ac.uk/isg/tools/KARMA/

solves Steps 2 and 3; that is, given q and the model computed in Step 1, it returns all certain answers to q w.r.t. the input $\mathcal{ELHO}^r_\bot$ knowledge base.

The computation of the datalog program from a $\mathcal{ELHO}^r_\bot$ knowledge base in Step 1 relies on a form of Skolemisation where existentially quantified variables are mapped to fresh constants (instead of functional terms).

Definition 7.6. For each rule r of the form (2.1)

$$\forall \mathbf{x} \forall \mathbf{y} (\beta_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \beta_n(\mathbf{x}, \mathbf{y}) \rightarrow \bigvee_{i=1}^{m} \exists \mathbf{z}_i \varphi_i(\mathbf{x}, \mathbf{z}_i))$$

and each existentially quantified variable z_{ij}, let o^r_{ij} be a constant globally unique for r and z_{ij}, and let $\theta_{\mathsf{c\text{-}sk}}$ be the substitution such that $\theta_{\mathsf{sk}}(z_{ij}) = o^r_{ij}$ for each $z_{ij} \in \mathbf{z}_i$. The *c-Skolemisation* $\mathsf{c\text{-}sk}(r)$ of r is given as follows:

$$\beta_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \beta_n(\mathbf{x}, \mathbf{y}) \rightarrow \bigvee_{i=1}^{m} \varphi_i(\mathbf{x}, \mathbf{z}_i)\theta_{\mathsf{c\text{-}sk}}.$$

Let $\mathcal{K}$ be a knowledge base, we define $\mathsf{c\text{-}sk}(\mathcal{K}) = \{\mathsf{c\text{-}sk}(r) \mid r \in \Sigma_\mathcal{H}\} \cup \mathcal{D}_\mathcal{H}$. ♢

Note that the application of c-Skolemisation to an $\mathcal{ELHO}^r_\bot$ rule of the form (EL1)–(EL3) in Definition 3.5 always results in a datalog rule. Note also that, in contrast to standard Skolemisation, c-Skolemisation is not a satisfiability or entailment preserving transformation. Let $\mathcal{H}$ be a knowledge base. There may be query answers w.r.t. $\mathsf{c\text{-}sk}(\mathcal{H})$ that are unsound w.r.t. $\mathcal{H}$. If $\mathcal{H}$ is $\mathcal{ELHO}^r_\bot$, however, it can be shown that $\mathcal{H}$ is satisfiable iff $\mathsf{c\text{-}sk}(\mathcal{H})$ is satisfiable (Stefanoni et al., 2013). We can obtain a lower bound on the query answers to q in $\mathcal{K}$ as follows:

- extract the subset $\Sigma^{EL}_\mathcal{K}$ of all $\mathcal{ELHO}^r_\bot$ rules in $\Sigma_\mathcal{K}$;
- compute the materialisation M of $\mathsf{c\text{-}sk}(\Sigma^{EL}_\mathcal{K}) \cup \mathcal{D}_\mathcal{K}$; and
- if $q = \bot$ then return unsatisfiable iff $\bot \in M$; otherwise, return $\mathsf{soundAnswers}(q, M)$.

By monotonicity of first-order logic, we have the set of sound answers computed from the $\mathcal{ELHO}^r_\bot$ subset is lower bound of query answers in $\mathcal{K}$.

Proposition 7.7. $\mathsf{cert}(q, \Sigma^{EL}_\mathcal{K} \cup \mathcal{D}_\mathcal{K}) \subseteq \mathsf{cert}(q, \mathcal{K})$.

Example 7.8. Consider again our running example. The $\mathcal{ELHO}^r_\bot$ fragment of $\mathcal{K}_{ex}$ consists of all facts (D1)–(D15) together with all rules except for (R4) and (R5). From fact (D12) and rule (R8) we deduce that *howler* eats a leaf, which must be a plant

by rule (R9). Hence *howler* is an answer to q_{ex}. This answer can be identified using the aforementioned steps. The c-Skolemisation of (R8a) leads to the datalog rule

$$\mathsf{Folivore}(x) \rightarrow \mathsf{eats}_L(x, o_3) \tag{R8aU}$$

The materialisation of the datalog program consisting of all facts and rule (R8aU) contains the fact $\mathsf{Plant}(o_3)$ and hence substitution $\{x \mapsto howler, y \mapsto o_3\}$ matches q_{ex} to the materialisation. This match is deemed sound by the filtration procedure. ♢

7.3 Aggregated Lower Bound

The techniques in this section can be seamlessly combined to obtain a lower bound L^q which is hopefully close to the actual set of certain answers. Given $\mathcal{K}$ and q, we proceed as follows:

1. Let $\Sigma_{\mathcal{K}}^{DD}$ be the subset of disjunctive datalog rules in $\Sigma_{\mathcal{K}}$. Construct the datalog program $\mathsf{shift}(\Sigma_{\mathcal{K}}^{DD})$ and compute the materialisation M_1^L of $\mathsf{shift}(\Sigma_{\mathcal{K}}^{DD}) \cup \mathcal{D}_{\mathcal{K}}$.

2. Let $\Sigma_{\mathcal{K}}^{EL}$ be the subset of $\mathcal{ELHO}_{\bot}^{r}$ rules in $\Sigma_{\mathcal{K}}$. Construct the datalog program $\mathsf{c\text{-}sk}(\Sigma_{\mathcal{K}}^{EL})$ and compute the materialisation M_2^L of $\mathsf{c\text{-}sk}(\Sigma_{\mathcal{K}}^{EL}) \cup M_1^L$ by reusing the materialisation from the previous step.

3. If $q = \bot$, then $L^q = \mathsf{cert}(q, M_2^L)$. Otherwise, $L^q = \mathsf{soundAnswers}(q, M_2^L)$.

Theorem 7.4 ensures that $\mathcal{K} \models \alpha$ for any atom $\alpha \in M_1^L$ in the signature of $\mathcal{K}$, and hence M_1^L can be used as the initial dataset for the second step. The properties of c-Skolemisation and filtration discussed in Section 7.2 then ensure that every answer in L^q is indeed a certain answer of q w.r.t. $\mathcal{K}$. Furthermore, if $\bot \in M_2^L$, then $\mathcal{K}$ is indeed unsatisfiable. Finally, note that the materialisation M_1^L obtained in the first step is "pipelined" into the second step; as a result, L^q is a (sometimes strict) superset of the answers we would obtain by simply computing the answers to q w.r.t. $\mathsf{shift}(\Sigma_{\mathcal{K}}^{DD}) \cup \mathcal{D}_{\mathcal{K}}$ and $\mathsf{c\text{-}sk}(\Sigma_{\mathcal{K}}^{EL}) \cup \mathcal{D}_{\mathcal{K}}$ independently and then unioning the results.

Example 7.9. For our running example $\mathcal{K}_{ex}$, the aggregated lower bound L_{ex} is

$$\{sheep, a\_hare, howler\}$$

where *sheep* follows from the datalog subset of $\mathcal{K}_{ex}$, *a_hare* follows from $\mathsf{shift}(\mathcal{K}_{ex})$, and *howler* follows from the $\mathcal{ELHO}_{\bot}^{r}$ subset of $\mathcal{K}_{ex}$. ♢

Chapter 8

Upper Bound Computation

In many practical cases the lower bound L^q described in Section 7.3 constitutes a rather precise approximation of the actual set of certain answers. Furthermore, it can also be computed very efficiently by resorting only to the datalog reasoner. The lower bound computation, however, gives no indication as to the accuracy of its answers: without a corresponding upper bound, every other possible answer remains a candidate answer, which needs to be either confirmed or discarded.

In this chapter, we describe our approach to efficiently computing an upper bound to the set of certain answers. If lower and upper bounds coincide, then the query has been fully answered; otherwise, the gap between lower and upper bounds not only provides a margin of error for the lower bound, but also narrows down the set of candidate answers whose verification may require more powerful computational techniques.

8.1 Strengthening the Knowledge Base

Our first step towards computing an upper bound will be to construct a (polynomial size) datalog knowledge base $\mathcal{K}'$ such that if $\mathcal{K}$ is unsatisfiable, then $\mathcal{K}'$ entails a nullary predicate $\bot_s$, and $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, \mathcal{K}')$ otherwise. Roughly speaking, such $\mathcal{K}'$, which we refer to as the *datalog strengthening* of $\mathcal{K}$, is obtained from $\mathcal{K}$ by

1. replacing $\bot$ by a fresh nullary predicate $\bot_s$ with no predefined meaning;
2. splitting the disjuncts in the head position into different datalog rules; and
3. Skolemising existentially quantified variables into constants as in Definition 7.6.

It is convenient for subsequent definitions and proofs to explicitly define the *splitting* of $\mathcal{K}$, written $\mathsf{split}(\mathcal{K})$, as the intermediate knowledge base resulting from Steps

1 and 2 above, which is both satisfiable and disjunction-free.[1] The datalog strengthening of $\mathcal{K}$ is then defined as the result of further applying Step 3 by replacing each existentially quantified rule in $\mathsf{split}(\mathcal{K})$ with its c-Skolemisation.

Definition 8.1. The *splitting* of a rule r of the form (2.1)

$$\forall \mathbf{x} \forall \mathbf{y} (\beta_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge \beta_n(\mathbf{x}, \mathbf{y}) \rightarrow \bigvee_{i=1}^{m} \exists \mathbf{z}_i \varphi_i(\mathbf{x}, \mathbf{z}_i))$$

is the following set of rules:

- if $\mathsf{head}(r) = \bot$, then $\mathsf{split}(r) = \{\beta_1 \wedge \cdots \wedge \beta_n \rightarrow \bot_s\}$, where $\bot_s$ is a fresh nullary predicate with no predefined meaning; and
- otherwise, $\mathsf{split}(r) = \{\, \beta_1 \wedge \cdots \wedge \beta_n \rightarrow \exists \mathbf{z}_j \varphi_j(\mathbf{x}, \mathbf{z}_j) \mid 1 \leq j \leq m \,\}$.

The splitting of $\mathcal{K} = \Sigma_\mathcal{K} \cup \mathcal{D}_\mathcal{K}$ is defined as $\mathsf{split}(\mathcal{K}) = \bigcup_{r \in \Sigma_\mathcal{K}} \mathsf{split}(r) \cup \mathcal{D}_\mathcal{K}$. Finally, the *datalog strengthening* of $\mathcal{K}$ is defined as $\mathsf{str}(\mathcal{K}) = \mathsf{c\text{-}sk}(\mathsf{split}(\mathcal{K}))$. ♢

Example 8.2. Consider our example knowledge base $\mathcal{K}_{ex}$. The splitting of $\mathcal{K}_{ex}$ is obtained by replacing rule (R5) with rules (R5Ua) and (R5Ub), and rule (R3) with (R3U).

$$\mathsf{Mammal}(x) \rightarrow \mathsf{Herbivore}(x) \tag{R5Ua}$$

$$\mathsf{Mammal}(x) \rightarrow \mathsf{MeatEater}(x) \tag{R5Ub}$$

$$\mathsf{Folivore}(x) \wedge \mathsf{MeatEater}(x) \rightarrow \bot_s \tag{R3U}$$

Finally, $\mathsf{str}(\mathcal{K})$ is obtained by further replacing the existentially quantified rules (R6a), (R7) with the following rules (R6aU), (R7U)

$$\mathsf{MeatEater}(x) \rightarrow \mathsf{eats}_H(x, o_1) \tag{R6aU}$$

$$\mathsf{Mammal}(x) \rightarrow \mathsf{eats}(x, o_2) \tag{R7U}$$

and replacing rule (R8a) with rule (R8aU) given in Example 7.8. ♢

Note that if $\mathcal{K}$ does not contain rules with $\bot$ in the head, then $\mathsf{str}(\mathcal{K})$ logically entails $\mathcal{K}$: splitting amounts to turning disjunctions in the head of rules into conjunctions, while c-Skolemisation restricts the possible values of existentially quantified variables to fixed constants. Thus, $\mathsf{cert}(q, \mathsf{str}(\mathcal{K}))$ constitutes an upper bound to $\mathsf{cert}(q, \mathcal{K})$, which is formally stated in the following proposition .

[1] The splitting of a knowledge base is a different notation from the "splitting of logic programs" in existing literature (Lifschitz and Turner, 1994).

Proposition 8.3. *if $\mathcal{K}$ is satisfiable, then we have* $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, \mathsf{str}(\mathcal{K}))$.

This is, however, no longer the case if $\mathcal{K}$ is unsatisfiable. In this case, $\mathsf{cert}(q, \mathcal{K})$ contains all substitutions from $\mathbf{x}$ to constants in $\mathcal{K}$, whereas $\mathsf{str}(\mathcal{K}))$ is always satisfiable since $\bot$ is replaced with an ordinary predicate $\bot_s$ without a predefined meaning. So, in this special case, $\mathsf{cert}(q, \mathsf{str}(\mathcal{K}))$ is a subset of $\mathsf{cert}(q, \mathcal{K})$. The rationale behind this replacement is to provide a meaningful upper bound even in cases where splitting disjunctions and c-Skolemising existentially quantified variables would make the strengthened knowledge base unsatisfiable. The example below provides some intuition for this rationale.

Example 8.4. Consider the datalog strengthening $\mathcal{K}'_{ex} = \mathsf{str}(\mathcal{K}_{ex})$ of our example knowledge base. Since *howler* is a Mammal, we have by rule (R5Ub) that it is also a $\mathsf{MeatEater}$. But then, since $\mathsf{Folivore}(howler)$ is a fact in $\mathcal{K}_{ex}$ we can derive $\bot_s$ using rule (R3U). Note that, had we not replaced the falsehood predicate $\bot$ with $\bot_s$, the datalog strengthening of $\mathcal{K}_{ex}$ would be unsatisfiable, in which case no meaningful upper bound could be obtained for any query. ♢

We next show that the datalog strengthening $\mathsf{str}(\mathcal{K})$ can be exploited to compute a meaningful upper bound for any input query, despite the fact that $\bot$ is stripped of its built-in semantics in first-order logic. The following lemma establishes the key property of the splitting transformation in Definition 8.1: if a ground clause $\varphi = \alpha_1 \vee \cdots \vee \alpha_n$ is derivable from $\mathcal{K}$ via hyperresolution, then the Skolem chase of $\mathsf{split}(\mathcal{K})$ contains every atom α_i for $1 \leq i \leq n$.

Lemma 8.5. *Let $\rho = (T, \lambda)$ be a hyperresolution derivation from $\mathcal{K}$ and let $\mathcal{H} =$ $\mathsf{split}(\mathcal{K})$. Then, for every node $v \in T$ and ground atom α occurring in $\lambda(v)$, we have that $\alpha \in \mathsf{skChase}(\mathcal{H})$.*

Proof. We prove the claim by bottom-up induction on the structure of T.

Base case: If v is a leaf in T, then $\lambda(v) \in \mathcal{D}_{\mathcal{K}}$. Since $\mathcal{D}_{\mathcal{K}} \subseteq \mathcal{H}$ we have $\alpha \in \mathsf{skChase}(\mathcal{H})$.

Inductive step: Assume that the induction hypothesis holds for all children $v_1, \ldots, v_n$ of a node u in T, we next prove that the lemma holds for u. Let r be the rule in $\mathcal{K}$ such that $\mathsf{sk}(r)$ is the main premise in the hyperresolution step to obtain $\lambda(u)$ with MGU σ, i.e. $\lambda(u) = \psi\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is the hyperresolvent of $\lambda(v_i) = \beta_i\sigma \vee \chi_i$ for $1 \leq i \leq n$ and $\mathsf{sk}(r) = \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \psi$. By the induction hypothesis, all the disjuncts in each χ_i are in $\mathsf{skChase}(\mathcal{H})$, so we only need to show the claim for

each disjunct in $\psi\sigma$. We distinguish the following cases depending on the form of the normalised rule r.

- If r is of the form ($\bot$-Norm), $\psi\sigma$ is empty. So the claim holds vacuously.
- If r is of the form ($\exists$-Norm), then $\psi = \gamma_1\theta_{\mathsf{sk}}$. By the induction hypothesis, each $\beta_i\sigma$ is in $\mathsf{skChase}(\mathcal{H})$, and since $\mathsf{split}(r) = r$ and hence $r \in \mathcal{H}$, we obtain $\gamma_1\theta_{\mathsf{sk}}\sigma \in \mathsf{skChase}(\mathcal{H})$.
- If r is of the form ($\vee$-Norm), then $\psi = \gamma_1 \vee \cdots \vee \gamma_m$. By induction hypothesis, each $\beta_i\sigma$ is in $\mathsf{skChase}(\mathcal{H})$, and for each $1 \leq i \leq m$, since the rule $\beta_1 \wedge \cdots \wedge \beta_n \rightarrow \gamma_i$ is in $\mathcal{H}$, we obtain that each atom $\gamma_i\sigma$ is also in $\mathsf{skChase}(\mathcal{H})$, as required. □

We can now exploit the completeness of hyperresolution to show that $\mathsf{split}(\mathcal{K})$ satisfies the required properties. Furthermore, the fact that $\mathsf{str}(\mathcal{K}) \models \mathsf{split}(\mathcal{K})$ immediately implies that $\mathsf{str}(\mathcal{K})$ satisfies those properties as well and hence it can be exploited to compute upper bound query answers.

Theorem 8.6. *The following properties hold for $\mathcal{H} = \mathsf{split}(\mathcal{K})$ as well as for $\mathcal{H} = \mathsf{str}(\mathcal{K})$: (i) $\mathsf{cert}(\bot, \mathcal{K}) \subseteq \mathsf{cert}(\bot_s, \mathcal{H})$, i.e. if $\mathcal{K}$ is unsatisfiable, then $\mathcal{H} \models \bot_s$; and (ii) if $\mathcal{K}$ is satisfiable then $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, \mathcal{H})$.*

Proof. We first show that properties (i) and (ii) hold for $\mathcal{H} = \mathsf{split}(\mathcal{K})$. If $\mathcal{K}$ is unsatisfiable, then there is a hyperresolution derivation of the empty clause from $\mathcal{K}$. Thus, there must exist a rule r of the form ($\bot$-Norm) in $\Sigma_{\mathcal{K}}$ and a substitution σ such that each atom $\beta_i\sigma$ for $1 \leq i \leq n$ is also derivable from $\mathcal{K}$. But then, by Lemma 8.5 we have that $\beta_i\sigma \in \mathsf{skChase}(\mathcal{H})$. Since $\mathcal{H}$ contains the rule $\beta_1 \wedge \cdots \wedge \beta_n \rightarrow \bot_s$ we have $\bot_s \in \mathsf{skChase}(\mathcal{H})$ and $\mathcal{H} \models \bot_s$, as required. Assume now that $\mathcal{K}$ is satisfiable. If $\mathsf{cert}(q, \mathcal{K}) = \emptyset$, $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, \mathcal{H})$ holds trivially; otherwise let σ be a certain answer to q w.r.t. $\mathcal{K}$. So $\mathcal{K} \models q\sigma$ and hence $\mathcal{K} \cup \mathcal{R}_q \models P_q(\mathbf{x})\sigma$. Since $\mathsf{cert}(\bot, \mathcal{K}) = \emptyset$, we have $q \neq \bot$. Using the completeness of hyperresolution and Lemma 8.5 we obtain that $P_q(\mathbf{x})\sigma$ is in the chase of $\mathcal{K} \cup \mathcal{R}_q$. But then, the aforementioned splitting also entails $P_q(\mathbf{x})\sigma$ and since $\mathsf{split}(\mathcal{K} \cup \mathcal{R}_q) = \mathcal{H} \cup \mathcal{R}_q$ we have $\sigma \in \mathsf{cert}(q, \mathcal{H})$, as required. Finally, properties *(i)* and *(ii)* hold for $\mathsf{str}(\mathcal{K})$ as a direct consequence of the fact that $\mathsf{str}(\mathcal{K}) \models \mathsf{split}(\mathcal{K})$. □

Example 8.7. Figure 8.1 depicts the materialisation of $\mathsf{str}(\mathcal{K}_{ex})$, where edges for predicates introduced during the normalisation are ignored and all edges in the figure represent the binary predicate eats. Explicit facts in $\mathcal{K}_{ex}$ are depicted in black; implicit

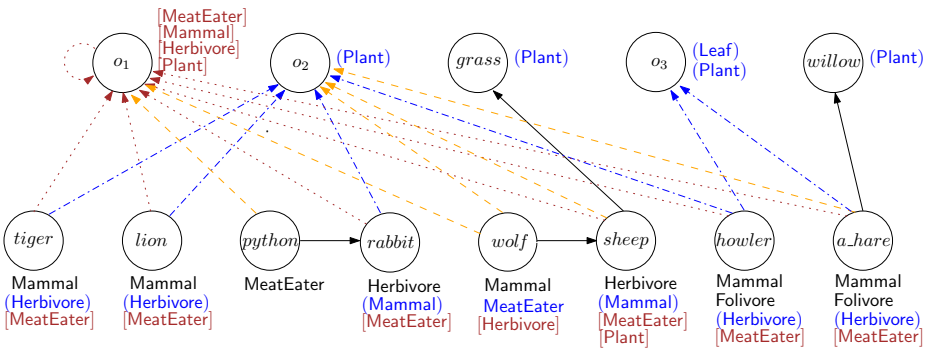


Figure 8.1: Materialisation of datalog strengthening of $\mathcal{K}_{ex}$

facts are depicted using different colours to facilitate the subsequent illustration of further refinements of the materialisation that will allow us to tighten the upper bound. We obtain the following upper bound of $\mathsf{cert}(q_{ex}, \mathcal{K}_{ex})$ by evaluating q_{ex} against the materialisation:

$$\mathsf{cert}(q_{ex}, \mathsf{str}(\mathcal{K}_{ex})) = \{tiger, lion, python, rabbit, wolf, sheep, howler, a\_hare, o_1\}$$

As already mentioned, $\mathsf{str}(\mathcal{K}_{ex}) \models \bot_s$; however, the obtained upper bound is still meaningful since it does not contain all possible answers in $\mathcal{K}_{ex}$, such as *grass* or *willow*. Please note that o_1 is a certain answer to q_{ex} w.r.t. $\mathsf{str}(\mathcal{K}_{ex})$; however, constant o_1 is not in $\mathsf{Sig}(\mathcal{K}_{ex})$ and hence it is not a possible answer to q_{ex} w.r.t. $\mathcal{K}$. ♢

8.2 Refined Handling for Existential Rules

The upper bound obtained from $\mathsf{str}(\mathcal{K})$ can be rather coarse-grained in practice: as discussed in Example 8.7, *python*, *tiger*, *lion* and *wolf* are contained in the upper bound, where none of them is a certain answer to q_{ex}. In this section, we show how to refine the upper bound by restricting the application of c-Skolemisation to existential rules. Instead of computing the upper bound of q by constructing the strengthened knowledge base $\mathsf{str}(\mathcal{K})$ and then evaluating q over (the materialisation of) $\mathsf{str}(\mathcal{K})$, we proceed as follows.

1. Apply to $\mathcal{K}$ a variant of the Skolem chase, which we refer to as the *c-chase* by first splitting the disjuncts occurring in head position into different rules.

2. Apply Skolem chasing on $\mathsf{split}(\mathcal{K})$ with the following modifications: *(i)* similarly to the *restricted chase* (Calì et al., 2013), existential rules are applied only when the rule head is not already satisfied; and

   (ii) rather than Skolemising existentially quantified variables (using a functional

term) whenever an existential rule is applied, we resort to c-Skolemisation instead. Due to the latter modification, the *c-chase* does not compute the least Herbrand Model of $\mathsf{split}(\mathcal{K})$, but rather just *some* model of $\mathsf{split}(\mathcal{K})$.

3. Evaluate q over the result of the aforementioned chase, thus obtaining an upper bound to the certain answers of q w.r.t. $\mathsf{split}(\mathcal{K})$, and thus also w.r.t. $\mathcal{K}$.

The following example motivates the practical advantages of this approach.

Example 8.8. Consider again the materialisation of $\mathsf{str}(\mathcal{K}_{ex})$ in Figure 8.1. As already mentioned, *python* is returned as an upper bound answer since q_{ex} is mapped to the materialisation by substitution $\{x \mapsto python, y \mapsto o_1\}$. The fact $\mathsf{eats}(python, o_1)$ is obtained from $\mathsf{eats}_H(python, o_1)$, which is included in the materialisation to satisfy the c-Skolemised rule (R6aU) in $\mathsf{str}(\mathcal{K}_{ex})$, and also the existentially quantified rule (R6a) in $\mathcal{K}_{ex}$. In the case of *python*, however, rule (R6a) in $\mathcal{K}_{ex}$ is already satisfied by the fact $\mathsf{eats}_H(python, rabbit)$, which is derived from $\mathsf{eats}(python, rabbit)$ and $\mathsf{Herbivore}(rabbit)$ in the dataset and rule (R6b). Please note that rule (R6b) is of the form (2.6) in the normalisation of (R6). Rule (R6b) ensures that if (R6) is satisfied for any substitution, then (R6a) is also satisfied for the same substitution. To obtain an upper bound it suffices to construct a model of $\mathcal{K}_{ex}$ (rather than a model of $\mathsf{str}(\mathcal{K}_{ex})$); thus, we can prevent the application of rule (R6aU) on $\{x \mapsto python\}$ during the chase, and thus dispense with $\mathsf{eats}(python, o_1)$ in the materialisation. ♢

We are now ready to define the *c-chase* formally.

Definition 8.9. Let $\mathcal{H} = \mathsf{split}(\mathcal{K})$, let $\Sigma^d_{\mathcal{H}}$ be the subset of datalog rules in $\Sigma_{\mathcal{H}}$, and $\Sigma^e_{\mathcal{H}} = \Sigma_{\mathcal{H}} \setminus \Sigma^d_{\mathcal{H}}$. The *c-chase sequence* of $\mathcal{K}$ is the sequence of datasets $\{\mathcal{B}^i\}_{i \geq 0}$, where $\mathcal{B}^0 = \mathcal{D}_{\mathcal{H}}$ (i.e. $\mathcal{B}^0 = \mathcal{D}_{\mathcal{K}}$), and $\mathcal{B}^{i+1}$ is inductively defined as given next. Let $\mathcal{S}^{i+1}_d$ and $\mathcal{S}^{i+1}_e$ be defined as follows:

$$\mathcal{S}^{i+1}_d = \{\mathsf{head}(r)\sigma \mid r \in \Sigma^d_{\mathcal{H}}, \mathcal{B}^i \models \mathsf{body}(r)\sigma \text{ and } \mathcal{B}^i \not\models \mathsf{head}(r)\}$$
$$\mathcal{S}^{i+1}_e = \{\mathsf{head}(\mathsf{c\text{-}sk}(r))\sigma \mid r \in \Sigma^e_{\mathcal{H}}, \mathcal{B}^i \models \mathsf{body}(r)\sigma \text{ and } \mathcal{B}^i \not\models \mathsf{head}(r)\}$$

Then, $\mathcal{B}^{i+1} = \mathcal{B}^i \cup \mathcal{S}^{i+1}_d$ if $\mathcal{S}^{i+1}_d \neq \emptyset$, and $\mathcal{B}^{i+1} = \mathcal{B}^i \cup \mathcal{S}^{i+1}_e$ otherwise. Finally, we define the *c-chase* of $\mathcal{K}$ as $\mathsf{c\text{-}Chase}_{\mathcal{K}} = \bigcup_{i \geq 0}\{\mathcal{B}^i\}$. ♢

Note that the the *c-chase* of $\mathcal{K}$ is a finite set since the only terms that can occur in it are constants from $\mathsf{c\text{-}sk}(\mathsf{split}(\mathcal{K}))$.

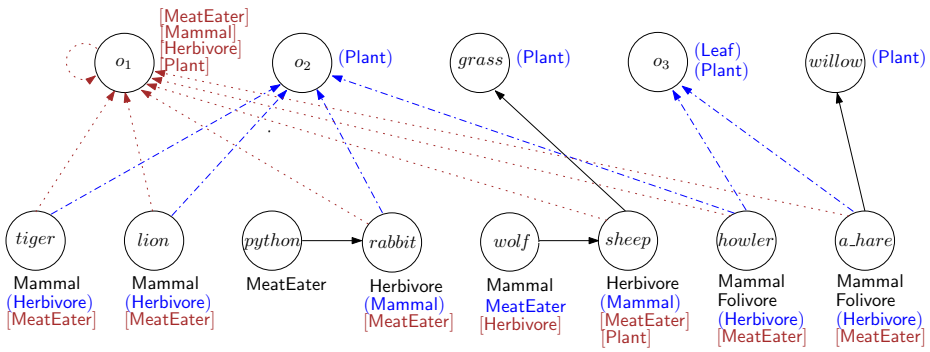


Figure 8.2: The c-chase of $\mathcal{K}_{ex}$

Example 8.10. The c-chase of $\mathcal{K}_{ex}$ is depicted in Figure 8.2. This materialisation is a strict subset of that in Figure 8.1, where the (orange-coloured) binary facts presented by dashed lines are no longer derived. Consequently, *python* is no longer derived as an answer to q_{ex}. ♢

The relevant properties of the c-chase are summarised in the following lemma.

Theorem 8.11. *The following properties hold: (i)* $\mathsf{cert}(\bot, \mathcal{K}) \subseteq \mathsf{cert}(\bot_s, \mathsf{c\text{-}Chase}_{\mathcal{K}})$*, i.e. if* $\mathcal{K}$ *is unsatisfiable, then* $\bot_s \in \mathsf{c\text{-}Chase}_{\mathcal{K}}$*; (ii) if* $\mathcal{K}$ *is satisfiable,* $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, \mathsf{c\text{-}Chase}_{\mathcal{K}})$*.*

Proof. We first prove that $\mathsf{c\text{-}Chase}_{\mathcal{K}}$ is a model of $\mathsf{split}(\mathcal{K})$. Since $\mathcal{D}_{\mathcal{K}} \subseteq \mathsf{c\text{-}Chase}_{\mathcal{K}}$ it is clear that it satisfies all facts in $\mathsf{split}(\mathcal{K})$. Let $r \in \mathsf{split}(\mathcal{K})$; we distinguish two cases:

- The rule r is datalog. If $\mathsf{c\text{-}Chase}_{\mathcal{K}} \models \mathsf{body}(r)\sigma$ for some substitution σ the definition of c-chase ensures that $\mathsf{head}(r)\sigma \in \mathsf{c\text{-}Chase}_{\mathcal{K}}$ and hence the rule is satisfied.

- Otherwise, r is of the form ($\exists$-`Norm`). If $\mathsf{c\text{-}Chase}_{\mathcal{K}} \models \mathsf{body}(r)\sigma$ for some substitution σ the definition of $\mathsf{c\text{-}Chase}_{\mathcal{H}}$ ensures that $\mathsf{head}(\mathsf{c\text{-}sk}(r))\sigma \in \mathsf{c\text{-}Chase}_{\mathcal{K}}$; thus, $\mathsf{c\text{-}Chase}_{\mathcal{K}} \models \mathsf{head}(r)\sigma$ and hence the rule is satisfied.

We now show the contrapositive of the first property. Assume that $\bot_s \notin \mathsf{c\text{-}Chase}_{\mathcal{K}}$. Because $\mathsf{c\text{-}Chase}_{\mathcal{K}}$ is a model of $\mathsf{split}(\mathcal{K})$, we have $\mathsf{split}(\mathcal{K}) \not\models \bot_s$ and hence $\mathcal{K}$ is satisfiable by Theorem 8.6. Finally, assume that $\mathcal{K}$ is satisfiable. If $\mathsf{cert}(q, \mathcal{K}) = \emptyset$, $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, \mathcal{H})$ holds trivially; otherwise let σ be a certain answer to q w.r.t. $\mathcal{K}$. By Theorem 8.6, we obtain $\sigma \in \mathsf{cert}(q, \mathsf{split}(\mathcal{K}))$. Because $\mathsf{c\text{-}Chase}_{\mathcal{K}} \models \mathsf{split}(\mathcal{K})$, we have $\sigma \in \mathsf{cert}(q, \mathsf{c\text{-}Chase}_{\mathcal{K}})$. □

8.3 Refined Handling for Disjunctive Rules

Although the technique described in the previous section can be quite effective in practice, its main limitation is that in $\mathsf{split}(\mathcal{K})$ all disjunctions in the heads of rules in $\mathcal{K}$ have effectively been turned into conjunctions. In this section, we show how to refine the upper bound by exploiting an extension of c-chase that uses a similar approach to deal with disjunctive rules as well as existential rules.

Specifically, we extend c-chase to deal with disjunctive rules r of the form ($\vee$-Norm) such that *(i)* r is applied only when none of the disjuncts in the head of the rule is already satisfied; and *(ii)* when r is applied, only one of the disjuncts is included in the chase (rather than all of them). In order to avoid non-determinism during chase expansion and reduce the computational cost, disjuncts are selected deterministically by means of an (efficiently implementable) *choice function*.

Example 8.12. Consider a knowledge base $\mathcal{H}$ consisting of the following axioms.

$$A(x) \to B(x) \vee C(x) \quad C(x) \wedge E(x) \to \bot \quad A(a) \quad E(a)$$

Assume that a choice function choose $C(a)$ over $B(a) \vee C(a)$. Then $\{A(a), E(a), C(a)\}$ is the resulted dataset by applying $A(x) \to B(x) \vee C(x)$ on $x \mapsto a$. Finally, we end up with the dataset $\{A(a), E(a), C(a), \bot_s\}$ that satisfies the knowledge base obtained from $\mathcal{H}$ by replacing $\bot$ with $\bot_s$. In this case, the answer $\{x \mapsto a\}$ to the query $q(x) = B(x)$ is missing if we evaluate the query against the resulted dataset. $\diamondsuit$

Therefore, besides efficiency, one of the criteria to design a choice function is to avoid deriving $\bot_s$ in the computed dataset.

Example 8.13. Consider again our running example. First observe that *wolf* is an answer to q_{ex} w.r.t. the c-chase of $\mathcal{K}_{ex}$ shown in Figure 8.2. Indeed, $\mathsf{Herbivore}(wolf)$ is derived from $\mathsf{Mammal}(wolf)$ and the rules in the split of (R5); thus, $\mathsf{Plant}(sheep)$ is also derived using rule (R4). Note, however, that *wolf* is a spurious answer: given that $\mathsf{MeatEater}(wolf)$ is an explicit fact in $\mathcal{K}_{ex}$, rule (R5) is already satisfied for *wolf* and hence we can dispense with fact $\mathsf{Herbivore}(wolf)$ in the materialisation.

Finally, since our goal is to construct a model of $\mathcal{K}_{ex}$ it is reasonable to pick disjuncts whose predicate is unrelated to $\bot$ in $\mathcal{K}_{ex}$. Since $\bot$ depends only on MeatEater and Folivore (by rule (R3)), it makes sense to include a fact $\mathsf{Herbivore}(b)$ in the materialisation whenever the disjunctive rule (R5) is applied to a constant b. $\diamondsuit$

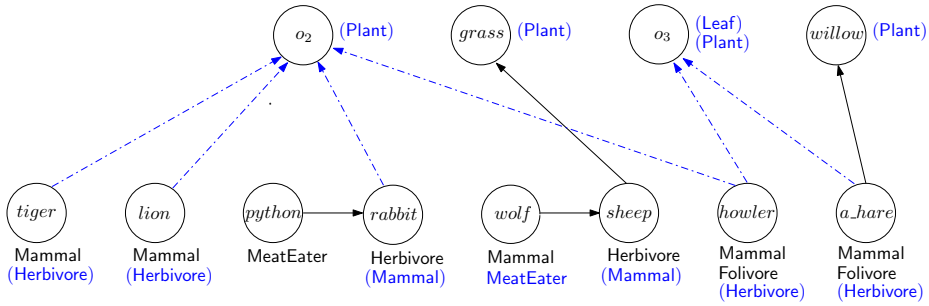


Figure 8.3: The c-chase^f of $\mathcal{K}_{ex}$

For further details we refer the reader to Section 11, where the specific choice function implemented in PAGOdA is described.

We can now define our extended notion of c-chase, where an efficiently implementable choice function is given as an additional parameter.

Definition 8.14. Let $\mathcal{H}$ be the knowledge base obtained from $\mathcal{K}$ by replacing $\bot$ with the nullary predicate $\bot_s$, let $\Sigma^d_{\mathcal{H}}$ be the set of all datalog rules in $\Sigma_{\mathcal{H}}$, and let $\Sigma^n_{\mathcal{H}} = \Sigma_{\mathcal{H}} \setminus \Sigma^d_{\mathcal{H}}$. Furthermore, let f be a polynomially computable *choice function* that given a positive ground clause χ and a dataset returns a disjunct in χ. The *c-chase* sequence of $\mathcal{K}$ w.r.t. f is the sequence of datasets $\{\mathcal{B}^i\}_{i\geq 0}$, where $\mathcal{B}^0 = \mathcal{D}_{\mathcal{H}}$ (i.e. $\mathcal{B}^0 = \mathcal{D}_{\mathcal{K}}$), and $\mathcal{B}^{i+1}$ is defined as given next. Let $\mathcal{S}^{i+1}_d$ and $\mathcal{S}^{i+1}_n$ be as follows:

$$\mathcal{S}^{i+1}_d = \{\mathsf{head}(r)\sigma \mid r \in \Sigma^d_{\mathcal{H}}, \mathcal{B}^i \models \mathsf{body}(r)\sigma \text{ and } \mathcal{B}^i \not\models \mathsf{head}(r)\}$$
$$\mathcal{S}^{i+1}_n = \{f(\mathsf{head}(\mathsf{c\text{-}sk}(r))\sigma, \mathcal{B}^i) \mid r \in \Sigma^n_{\mathcal{H}}, \mathcal{B}^i \models \mathsf{body}(r)\sigma \text{ and } \mathcal{B}^i \not\models \mathsf{head}(r)\}$$

Then, $\mathcal{B}^{i+1} = \mathcal{B}^i \cup \mathcal{S}^{i+1}_d$ if $\mathcal{S}^{i+1}_d \neq \emptyset$, and $\mathcal{B}^{i+1} = \mathcal{B}^i \cup \mathcal{S}^{i+1}_n$ otherwise. Finally, we define the *c-chase* of $\mathcal{K}$ w.r.t. f as $\mathsf{c\text{-}Chase}^f_{\mathcal{K}} = \bigcup_{i\geq 0}\{\mathcal{B}^i\}$. ♢

Example 8.15. Consider the aforementioned choice function f that picks $\mathsf{Herbivore}(b)$ whenever rule (R5) is applied to a fact $\mathsf{Mammal}(b)$. Figure 8.3 depicts the facts in $\mathsf{c\text{-}Chase}^f_{\mathcal{K}_{ex}}$. It can be observed that $\mathsf{c\text{-}Chase}^f_{\mathcal{K}_{ex}}$ is a strict subset of the materialisation in Figure 8.2, where the (brown-coloured) facts presented in brackets or by dotted lines are no longer derived. We can see that *wolf* is not an answer to q_{ex} w.r.t. $\mathsf{c\text{-}Chase}^f_{\mathcal{K}_{ex}}$ and hence it can be identified as spurious. Furthermore, the nullary predicate $\bot_s$ has not been derived and hence we can determine that $\mathcal{K}_{ex}$ is satisfiable. ♢

The relevant properties of this variant of the c-chase are as follows.

Theorem 8.16. *Let f be a choice function as required in Definition 8.14. If $\bot_s \notin \mathsf{c\text{-}Chase}^f_{\mathcal{K}}$, then $\mathsf{c\text{-}Chase}^f_{\mathcal{K}}$ is a model of $\mathcal{K}$ and $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, \mathsf{c\text{-}Chase}^f_{\mathcal{K}})$.*

Proof. The dataset $\mathcal{D}_\mathcal{K}$ is contained in $\mathsf{c\text{-}Chase}^f_\mathcal{K}$, so it suffices to show that $\mathsf{c\text{-}Chase}^f_\mathcal{K}$ satisfies each rule $r \in \mathcal{K}$. We distinguish the following cases:

- r is of the form ($\bot$-Norm). Since $\bot_s \notin \mathsf{c\text{-}Chase}^f_\mathcal{K}$, there cannot exist a substitution σ such that $\mathsf{c\text{-}Chase}^f_\mathcal{K} \models \mathsf{body}(r)\sigma$ and hence $\mathsf{c\text{-}Chase}^f_\mathcal{K}$ satisfies r vacuously.
- r is of the form ($\exists$-Norm). Pick a substitution σ such that $\mathsf{c\text{-}Chase}^f_\mathcal{K} \models \mathsf{body}(r)\sigma$. The definition of $\mathsf{c\text{-}Chase}^f_\mathcal{K}$ ensures that $\mathsf{head}(\mathsf{c\text{-}sk}(r))\sigma \in \mathsf{c\text{-}Chase}^f_\mathcal{K}$ and hence $\mathsf{c\text{-}Chase}^f_\mathcal{K}$ satisfies r.
- r is of the form ($\vee$-Norm). Pick a substitution σ such that $\mathsf{c\text{-}Chase}^f_\mathcal{K} \models \mathsf{body}(r)\sigma$. By the definition of $\mathsf{c\text{-}Chase}^f_\mathcal{K}$, we have $f(\mathsf{head}(\mathsf{c\text{-}sk}(r)), S^i_n)\sigma \in \mathsf{c\text{-}Chase}^f_\mathcal{K}$ for some set of atoms S^i_n in the chase sequence, and then $\mathsf{c\text{-}Chase}^f_\mathcal{K}$ satisfies r.

If $q = \bot$, then $\mathsf{cert}(q, \mathcal{K}) = \emptyset$ and $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, \mathsf{c\text{-}Chase}^f_\mathcal{K})$ holds trivially; otherwise, $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, \mathsf{c\text{-}Chase}^f_\mathcal{K})$ follows directly from the fact that $\mathsf{c\text{-}Chase}^f_\mathcal{K}$ is a model of $\mathcal{K}$. □

8.4 Combined Upper Bound

We have introduced three different techniques for computing upper bounds of $\mathsf{cert}(q, \mathcal{K})$.

1. Compute the materialisation M^U_1 of $\mathsf{str}(\mathcal{K})$, and evaluate q w.r.t. M^U_1 to obtain a set of possible answers U^q_1 to q w.r.t. $\mathcal{K}$ (cf. Section 8.1).
2. Compute the c-chase of $\mathcal{K}$, denoted by M^U_2, and evaluate q w.r.t. M^U_2 to obtain a set of possible answers U^q_2 to q w.r.t. $\mathcal{K}$ (cf. Section 8.2).
3. Fix a choice function f, compute the c-chase of $\mathcal{K}$ w.r.t. f, denoted by M^U_3, and evaluate q w.r.t. M^U_3 to obtain a set of possible answers U^q_3 to q w.r.t. $\mathcal{K}$ (cf. Section 8.3).

It can be trivially seen that U^q_2 and U^q_3 are at least as precise as U^q_1, i.e. $U^q_2 \subseteq U^q_1$ and $U^q_3 \subseteq U^q_1$. As shown in the following example, U^q_2 and U^q_3 are, however, incomparable.

Example 8.17. Consider a knowledge base $\mathcal{H}$ consisting of facts $A(a_1), R(a_1, b_1)$, $B(b_1)$, $A(a_2)$, $R(a_2, b_2)$, $B(b_2)$ and rules $B(x) \rightarrow C(x) \vee D(x)$, $R(x, y) \wedge C(y) \rightarrow S(x, y)$ and $A(x) \rightarrow \exists y S(x, y)$. Let o be the freshly introduced constant for $B(x) \rightarrow$

$\exists y S(x,y)$, and let f be a choice function that picks the disjunct $D(b_i)$ in every clause $C(b_i) \vee D(b_i)$. Then,

$$\begin{aligned}\mathsf{c\text{-}Chase}_{\mathcal{H}} &= \mathcal{D}_{\mathcal{H}} \cup \{C(b_1), D(b_1), S(a_1,b_1), C(b_2), D(b_2), S(a_2,b_2)\}, \text{ and}\\ \mathsf{c\text{-}Chase}^{f}_{\mathcal{H}} &= \mathcal{D}_{\mathcal{H}} \cup \{D(b_1), S(a_1,o), D(b_2), S(b_1,o)\}.\end{aligned}$$

For $q_1(x) = \exists y(S(x,y) \wedge C(y) \wedge D(y))$, the upper bound computed using the $\mathsf{c\text{-}Chase}_{\mathcal{H}}$ contains two additional answers a_1 and a_2 compared with that computed using $\mathsf{c\text{-}Chase}^{f}_{\mathcal{H}}$. But for $q_2(x_1,x_2) = \exists y(S(x_1,y) \wedge S(x_2,y))$, the upper bound computed using $\mathsf{c\text{-}Chase}^{f}_{\mathcal{H}}$ has additional answers (a_1,a_2) and (a_2,a_1) compared with that computed using $\mathsf{c\text{-}Chase}_{\mathcal{H}}$. ♢

There are, however, tradeoffs to be considered. Clearly, the upper bound U_1^q is the most convenient from an ease of implementation point of view: once $\mathsf{str}(\mathcal{K})$ has been constructed, the bound can be directly computed using an off-the-shelf datalog reasoner without modification. Furthermore, the upper bound U_3^q has an important shortcoming: it is of no use whenever $\bot_s$ is derived, as we will show in the following example.

Example 8.18. Consider a choice function g that picks $\mathsf{MeatEater}(a)$ for any disjunction of the form $\mathsf{Herbivore}(a) \vee \mathsf{MeatEater}(a)$. Then the c-chase of $\mathcal{K}_{ex}$ w.r.t. g will derive $\mathsf{MeatEater}(\mathit{howler})$ from the fact $\mathsf{Mammal}(\mathit{howler})$ and the disjunctive rule (R5). Using fact $\mathsf{Folivore}(\mathit{howler})$ and rule (R3U) it will then derive $\bot_s$. Thus we can see that, although *howler* is in $\mathsf{cert}(q_{ex}, \mathcal{K}_{ex})$, $\mathsf{Herbivore}(\mathit{howler})$ is not in the c-chase of $\mathcal{K}_{ex}$ w.r.t. g, and hence *howler* is not in the upper bound computed using it; this is in contrast to the other two upper bounds, where $\mathsf{Herbivore}(\mathit{howler})$ is in the materialisation of $\mathsf{str}(\mathcal{K}_{ex})$ and the c-chase of $\mathcal{K}_{ex}$, and hence *howler* is in the upper bound computed w.r.t. them. ♢

Therefore, if $\bot_s \notin \mathsf{c\text{-}Chase}^{f}_{\mathcal{K}}$, we can intersect U_2^q and U_3^q to compute a hopefully more precise upper bound; otherwise, we can simply use U_2^q.

Definition 8.19. The combined upper bound query answer U^q to q in $\mathcal{K}$ is formally defined as follows:

$$U^q = \begin{cases} U_2^{\bot_s} \cap U_3^{\bot_s} & \text{if } q = \bot; \\ U_2^q \cap U_3^q & \text{if } q \neq \bot \text{ and } \bot_s \notin \mathsf{c\text{-}Chase}^{f}_{\mathcal{K}}; \\ U_2^q & \text{otherwise.} \end{cases}$$ ♢

Example 8.20. The combined upper bound for q_{ex} in $\mathcal{K}_{ex}$ gives:

$$U_{ex} = \{tiger, lion, rabbit, sheep, howler, a\_hare\}.$$

If we compare this upper bound with the aggregated lower bound L_{ex} given in Example 7.9 we can identify a gap $G_{ex} = \{tiger, lion, rabbit\}$. ♢

The computation of both lower and upper bound is in polynomial time to the size of the dataset. This statement is straightforward for lower bound since the data complexity of conjunctive query answering in datalog is PTime. Regarding the upper bound, the computation is still polynomial because of the following two reasons.

- When applying existential rules, we use c-Skolemisation instead of the rule itself. So the number of freshly generated constants are bounded by the number of rules in the program.
- For disjunctive rules, the approach simply makes one choice between the disjuncts using a choice function without back-tracking. So it is treated "deterministically" for a given choice function.

So the query bounds computation only adds a polynomial delay to the fully-fledged reasoning, even when the bounds are trivial. Moreover, it is witnessed by our evaluation that the lower and upper bounds are very effective in pruning answers.

Example 8.21. Consider another query for our running example $q(x) = \mathsf{Plant}(x)$. It is clear that $\mathsf{Plant}(grass)$ is entailed by the datalog subset of $\mathcal{K}_{ex}$. Moreover, as shown in Example 7.2, *willow* is an answer in the lower bound as well. For this query, we have

$$L^q = \{grass, willow\}$$
$$U^q = \{o_2, grass, o_3, willow\}$$

Since it is trivial that o_2 and o_3 can not be certain answers of q in $\mathcal{K}_{ex}$, we can ignore both of them in U^q. Therefore, we have $\mathsf{cert}(q, \mathcal{K}_{ex}) = \{grass, willow\}$ after computing the lower and upper bounds.

Chapter 9

Reducing the Size of the Knowledge Base

In cases where there is a non-empty gap G^q between lower and upper bound (e.g., our running example) we need to verify whether each answer in G^q is spurious or not. Accomplishing this task using a fully-fledged reasoner can be computationally very expensive: verifying each answer in G^q typically involves at least one satisfiability test, which can be infeasible in practice for large-scale knowledge bases.

In this section we propose a technique for identifying a (typically small) subset $\mathcal{K}^q$ of the knowledge base $\mathcal{K}$ that are sufficient for verifying all answers in G^q (i.e. $\sigma \in \mathsf{cert}(q, \mathcal{K})$ iff $\sigma \in \mathsf{cert}(q, \mathcal{K}^q)$ for each $\sigma \in G^q$). The idea to restrict the dataset to a relevant subset has been exploited in some other contexts, e.g. for bounded-evaluable queries in database (Cao et al., 2014). It is essential that these subsets be, on the one hand, as small as possible and, on the other hand, efficiently computable. These requirements are in conflict: computing minimal-sized subsets can be as hard as answering the query, whereas subsets that can be easily computed may be almost as large as the initial knowledge base.

The main idea behind our approach is to construct a datalog knowledge base whose materialisation identifies all rules and facts in $\mathcal{K}^q$. Such a knowledge base is of size polynomial in the sizes of $\mathcal{K}$ and q and it does not include predicates of arity higher than those in $\mathcal{K}$ or q. In this way, subset computation can be fully delegated to the scalable datalog reasoner, hence addressing the efficiency requirement. The key property of $\mathcal{K}^q$, which ensures that it contains all the relevant information in $\mathcal{K}$, is the following: for each rule or fact $\alpha \notin \mathcal{K}^q$ we can show that α does not occur in any hyperresolution proof of $\square$ (resp. a gap answer in G^q) from $\mathcal{K} \cup \mathcal{R}_q$ for $q = \bot$ (resp. $q \neq \bot$). The completeness of hyperresolution then guarantees that all excluded facts and rules are indeed irrelevant.

9.1 Overview of the Approach

Let us motivate the main ideas behind our approach using our running example. Since $\bot_s$ has not been derived in $M_2^U \cap M_3^U$, we know that $\mathsf{cert}(\bot, \mathcal{K}_{ex}) = \emptyset$, and hence that $\mathcal{K}_{ex}$ is satisfiable (see Example 8.15). However, we still need to determine whether answers in $G_{ex} = \{tiger, lion, rabbit\}$ from the combined upper bound are in $\mathsf{cert}(q_{ex}, \mathcal{K}_{ex})$, i.e., if they are certain answers to q_{ex}.

We now sketch the construction of a datalog knowledge base $\mathsf{track}(\mathcal{K}_{ex}, q_{ex}, G_{ex})$ from which the subset of $\mathcal{K}_{ex}$ relevant to the answers in G_{ex} is derived. The key property of this knowledge base is that its materialisation "tracks" all the rules and facts that may participate in a hyperresolution proof of a gap answer and thus encodes the contents of the subset $\mathcal{K}^{q_{ex}}$. The relevant information is recorded using fresh predicates and constants:

- a fresh predicate P^R for each predicate P in $\mathcal{K}_{ex}$, the extension of which in the materialisation of $\mathsf{track}(\mathcal{K}_{ex}, q_{ex}, G_{ex})$ will give us the facts in the subset;
- a fresh constant d_r for each rule r in $\mathcal{K}_{ex}$ and a special unary predicate Rel, the extension of which in the materialisation of $\mathsf{track}(\mathcal{K}_{ex}, q_{ex}, G_{ex})$ will give us the rules in the subset.

The key step in the construction of this knowledge base is to "invert" each rule $r \in \mathcal{K}_{ex}$ into a set of datalog rules $\Delta(r)$ by *(i)* moving all head atoms of r into the body while replacing their predicates with the corresponding fresh ones (e.g., replace P with P^R); *(ii)* "copying" all the atoms that were originally in the body of r into the (now empty) head while replacing predicates with the corresponding fresh ones and adding the special atom $\mathsf{Rel}(d_r)$ as an additional conjunct; and *(iii)* eliminating the conjunction in the head of r by splitting r into multiple rules, one for each head conjunct.

Consider as a first example the datalog rule (R4) in $\mathcal{K}_{ex}$, which is inverted into the following rules:

$$\mathsf{Plant}^R(y) \wedge \mathsf{Herbivore}(x) \wedge \mathsf{eats}(x, y) \rightarrow \mathsf{Herbivore}^R(x) \tag{9.1}$$

$$\mathsf{Plant}^R(y) \wedge \mathsf{Herbivore}(x) \wedge \mathsf{eats}(x, y) \rightarrow \mathsf{eats}^R(x, y) \tag{9.2}$$

$$\mathsf{Plant}^R(y) \wedge \mathsf{Herbivore}(x) \wedge \mathsf{eats}(x, y) \rightarrow \mathsf{Rel}(\mathsf{d}_{R4}) \tag{9.3}$$

The head $\mathsf{Plant}(y)$ of (R4) has been moved to the body and predicate Plant replaced with Plant^R; the body $\mathsf{Herbivore}(x) \wedge \mathsf{eats}(x, y)$ has been "copied" into the head as

the conjunction $\mathsf{Herbivore}^R(x) \wedge \mathsf{eats}^R(x, y)$, and then conjoined with the special atom $\mathsf{Rel}(\mathsf{d}_{R4})$; and finally the head conjunction has been eliminated by splitting the rule into three separate rules.

These rules reflect the intuitive meaning of the freshly introduced predicates. If fact $\mathsf{Plant}^R(c)$ holds for some constant c, this means that fact $\mathsf{Plant}(c)$ may participate in a hyperresolution proof in $\mathcal{K}_{ex}$ of an answer in the gap. Additionally, if $\mathsf{Herbivore}(b)$ and $\mathsf{eats}(b, c)$ also hold for some b, then these facts and the rule (R4) could also participate in one such proof since $\mathsf{Plant}(c)$ is a hyperresolvent of $\mathsf{Herbivore}(b)$, $\mathsf{eats}(b, c)$ and rule (R4), which is recorded as facts $\mathsf{Herbivore}^R(b)$, $\mathsf{eats}^R(b, c)$, and $\mathsf{Rel}(\mathsf{d}_{R4})$. Thus, rules (9.1)–(9.3) faithfully "invert" hyperresolution steps involving rule (R4).

Similarly, the disjunctive rule (R5) is inverted into the following two rules:

$$\mathsf{Herbivore}^R(x) \wedge \mathsf{MeatEater}^R(x) \wedge \mathsf{Mammal}(x) \rightarrow \mathsf{Mammal}^R(x) \tag{9.4}$$

$$\mathsf{Herbivore}^R(x) \wedge \mathsf{MeatEater}^R(x) \wedge \mathsf{Mammal}(x) \rightarrow \mathsf{Rel}(\mathsf{d}_{R5}) \tag{9.5}$$

In this case, the disjunctive head $\mathsf{Herbivore}(x) \vee \mathsf{MeatEater}(x)$ of (R5) has been moved to the body as the conjunction $\mathsf{Herbivore}^R(x) \wedge \mathsf{MeatEater}^R(x)$ over the fresh predicates $\mathsf{Herbivore}^R$ and $\mathsf{MeatEater}^R$. If facts $\mathsf{Herbivore}^R(c)$ and $\mathsf{MeatEater}^R(c)$ hold for some c (which means that both facts $\mathsf{Herbivore}(c)$ and $\mathsf{MeatEater}(c)$ may participate in a relevant proof in $\mathcal{K}_{ex}$) and $\mathsf{Mammal}(c)$ holds, then we also deem fact $\mathsf{Mammal}(c)$ and rule (R5) relevant.

The situation is different when it comes to inverting an existentially quantified rules, in which case we no longer capture relevant hyperresolution steps in $\mathcal{K}_{ex}$ faithfully. Consider rule (R7), which is inverted as follows:

$$\mathsf{eats}^R(x, y) \wedge \mathsf{Mammal}(x) \rightarrow \mathsf{Mammal}^R(x) \tag{9.6}$$

$$\mathsf{eats}^R(x, y) \wedge \mathsf{Mammal}(x) \rightarrow \mathsf{Rel}(\mathsf{d}_{R7}) \tag{9.7}$$

In this case, the existentially quantified head $\exists y\, \mathsf{eats}(x, y)$ is moved to the body as the atom $\mathsf{eats}^R(x, y)$. If $\mathsf{eats}^R(b, c)$ holds for some b and c (and hence this fact may participate in a relevant proof), and $\mathsf{Mammal}(b)$ also holds, then we record both (R7) and $\mathsf{Mammal}(b)$ as relevant (the latter by means of the fact $\mathsf{Mammal}^R(b)$). The hyperresolvent of $\mathsf{Mammal}(b)$ and (R7) is an atom $\mathsf{eats}(b, t)$, with t a functional term over b, which may be unrelated to $\mathsf{eats}(b, c)$ and hence irrelevant to proving an answer in the gap.

In addition to inverting the rules in $\mathcal{K}_{ex}$, the construction of $\mathsf{track}(\mathcal{K}_{ex}, q_{ex}, G_{ex})$ also needs to take the query and gap answers into account. For this, we encode the

query $\mathsf{eats}(x, y) \wedge \mathsf{Plant}(y) \rightarrow P_{q_{ex}}(x)$ into the rules

$$P^R_{q_{ex}}(x) \wedge \mathsf{eats}(x, y) \wedge \mathsf{Plant}(y) \rightarrow \mathsf{eats}^R(x, y) \tag{9.8a}$$

$$P^R_{q_{ex}}(x) \wedge \mathsf{eats}(x, y) \wedge \mathsf{Plant}(y) \rightarrow \mathsf{Plant}^R(y) \tag{9.8b}$$

and add a fact $P^R_{q_{ex}}(c)$ for each $c \in G_{ex}$. These query-dependent rules are used to initialise the extension of the fresh predicates, which subsequently makes the other rules in $\mathsf{track}(\mathcal{K}_{ex}, q_{ex}, G_{ex})$ applicable.

The query answers in the gap stem from the upper bound; consequently, in order for rules (9.8a) and (9.8b) to be applicable the data in $\mathsf{track}(\mathcal{K}_{ex}, q_{ex}, G_{ex})$ is obtained from the upper bound materialisation of $\mathcal{K}_{ex}$. In the following section we show that it suffices to include all facts in the c-chase of $\mathcal{K}_{ex}$ in order to ensure that the computed subset will contain all the necessary facts and rules.

9.2 Subset Definition and Properties

We are now ready to formally define the datalog knowledge base used for subset computation as well as the corresponding relevant subset.

Definition 9.1. Let G be a set of possible answers to q, let Rel be a fresh unary predicate and let d_r be a fresh constant unique to each r in $\mathcal{K} \cup \mathcal{R}_q$. Furthermore, for each predicate P in $\mathcal{K} \cup \mathcal{R}_q$, let P^R be a fresh predicate of the same arity as P and, for an atom $\alpha = P(\mathbf{t})$, let α^R denote $P^R(\mathbf{t})$. For any normalised rule $r \in \mathcal{K} \cup \mathcal{R}_q$, let $\mathsf{move}(r)$ be the following conjunction of atoms:

- $P^R_\perp$ if r of the form ($\perp$-`Norm`);
- $\gamma^R_1(\mathbf{x}, \mathbf{z}_1)$ if r of the form ($\exists$-`Norm`); and
- $\gamma^R_1(\mathbf{x}) \wedge \cdots \wedge \gamma^R_m(\mathbf{x})$ if r of the form ($\vee$-`Norm`).

Then, $\Delta(r)$ is the following set of rules:

$$\Delta(r) = \{\mathsf{move}(r) \wedge \mathsf{body}(r) \rightarrow \mathsf{Rel}(\mathsf{d}_r)\} \cup \{\mathsf{move}(r) \wedge \mathsf{body}(r) \rightarrow \beta^R_k \mid \beta_k \text{ in } \mathsf{body}(r)\}.$$

The tracking knowledge base $\mathsf{track}(\mathcal{K}, q, G)$ is the smallest knowledge base containing

(i) all facts in the c-chase of $\mathcal{K}$;

(ii) all rules in $\bigcup_{r \in \mathcal{K} \cup \mathcal{R}_q} \Delta(r)$;

(iii) a fact $P_q^R(\mathbf{x})\sigma$ for each $\sigma \in G$;[1] and

(iv) a fact $P_\bot^R$ if $q \neq \bot$.

The subset of $\mathcal{K}$ relevant to q and G, denoted by $\mathcal{K}^{q,G}$, is the smallest knowledge base containing

- each rule $r \in \Sigma_\mathcal{K}$ such that $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{Rel}(\mathsf{d}_r)$; and
- each fact $\alpha \in \mathcal{D}_\mathcal{K}$ such that $\mathsf{track}(\mathcal{K}, q, G) \models \alpha^R$.

For brevity, we write $\mathcal{K}^q$ for the particular case where G is the set of gap answers $U_q \setminus L_q$ as defined in Section 7.3 and 8.4. ♢

Note that $\mathcal{K}^\bot$ is a subset of $\mathcal{K}^q$ since $\mathsf{track}(\mathcal{K}, \bot, G^\bot)$ is a subset of $\mathsf{track}(\mathcal{K}, q, G^q)$: in Definition 9.1, point *(i)* is the same for $\mathsf{track}(\mathcal{K}, \bot, G^\bot)$ and $\mathsf{track}(\mathcal{K}, q, G^q)$; furthermore, the set of rules from *(ii)* for $\mathsf{track}(\mathcal{K}, \bot, G^\bot)$ is a subset of that for $\mathsf{track}(\mathcal{K}, q, G^q)$ since $\mathcal{K} \cup \mathcal{R}_\bot \subseteq \mathcal{K} \cup \mathcal{R}_q$; finally, the fact $P_\bot^R$, which is included in $\mathsf{track}(\mathcal{K}, \bot, G^\bot)$ by point *(iii)*, also belongs to $\mathsf{track}(\mathcal{K}, q, G^q)$ by point *(iv)*.

Example 9.2. Consider again our running example, where $G_{ex} = \{tiger, lion, rabbit\}$. The subset of $\mathcal{K}_{ex}$ relevant to q_{ex} and G_{ex} consists of rules R2, R4, R5, R6, and R7 and facts D1, D2, D3, D5, D7, D9, and D11. ♢

We next give an example illustrating why facts in $\mathsf{track}(\mathcal{K}, q, G)$ (point *(i)* in Definition 9.1) are obtained from $\mathsf{c\text{-}Chase}_\mathcal{K}$ the materialisation underpinning the upper bound in Section 8.2, rather than $\mathsf{c\text{-}Chase}_\mathcal{K}^f$ in Section 8.3.

Example 9.3. Consider a query $q(x) = E(x)$ and a knowledge base $\mathcal{K}$ consisting of the following rules and facts.

$$A(x) \to B(x) \vee D(x) \qquad D(x) \to E(x)$$
$$B(x) \to E(x) \qquad A(a)$$

Let f be a function choosing $B(a)$ over $D(a)$, then $\mathsf{c\text{-}Chase}_\mathcal{K}^f = \{A(a), B(a), E(a)\}$ and constant a is an answer to $q(x)$ in the gap between the (empty) lower bound and the upper bound. Suppose that we were to define $\mathsf{track}(\mathcal{K}, q, G)$ as in Definition 9.1 but replacing the facts in point *(i)* with those in $\mathsf{c\text{-}Chase}_\mathcal{K}^f$. Since $D(a)$ does not hold in $\mathsf{c\text{-}Chase}_\mathcal{K}^f$ the corresponding subset will not contain the rule $D(x) \to E(x)$, which is essential to derive $E(a)$. ♢

[1] If P_q is of arity more than two and the underlying datalog reasoner supports predicates of arity no more than two, this item can be equivalently replaced by all rules in $\Delta(\varphi(\mathbf{x}, \mathbf{y}) \to P_q(\mathbf{x}))\sigma$ for each $\sigma \in G$.

In the following, we define a notation of query goal to facilitate proofs related to subset extraction. Let G be a set of possible answers to q, we define a set $\Upsilon(q, G)$ of *query goals* for q and G to facilitate proofs.

$$\Upsilon(q, G) = \begin{cases} \{\Box\} & \text{if } q = \bot; \\ \{P_q(\mathbf{x})\sigma \mid \sigma \in G\} & \text{otherwise.} \end{cases}$$

The key properties of computed subsets are established by the following theorem. We conclude this section with the theorem and its proof.

Theorem 9.4. *The following properties hold:*

(1) Assume that $L^\bot = \emptyset$. Then, $\mathcal{K}$ is unsatisfiable iff $\mathcal{K}^\bot$ is unsatisfiable.

(2) Let q be different from $\bot$ and let G be any non-empty set of possible answers to q w.r.t. $\mathcal{K}$. If $\mathcal{K}$ is satisfiable, then $\sigma \in \mathsf{cert}(q, \mathcal{K})$ iff $\sigma \in \mathsf{cert}(q, \mathcal{K}^{q,G})$ for every $\sigma \in G$.

Proof. The "if" direction of *(1)* and *(2)* follows directly from the monotonicity of first-order logic. The "only if" direction of both *(1)* and *(2)* follows from the completeness of hyperresolution and the following claim, which establishes that for any q and a non-empty G, $\mathcal{K}^{q,G}$ contains the support of all hyperresolution derivations of a query goal in $\Upsilon(q, G)$ from $\mathcal{K} \cup \mathcal{R}_q$ where

Claim (♣) If $\rho = (\lambda, T)$ is a hyperresolution derivation of a goal from $\Upsilon(q, G)$ from $\mathcal{K} \cup \mathcal{R}_q$, then $\mathsf{support}(\rho) \subseteq \mathcal{K}^{q,G}$.

To show the "only if" direction of *(1)*, assume that $\mathcal{K}$ is unsatisfiable. By Theorem 8.11, Theorem 8.16 and (8.19), we have $U^\bot \neq \emptyset$ and thus $G^\bot \neq \emptyset$. There exists a hyperresolution derivation ρ_1 of $\Box$ from $\mathcal{K}$. Since $\Upsilon(\bot, G^\bot) = \{\Box\}$, we know that $\mathsf{support}(\rho_1) \subseteq \mathcal{K}^\bot$ by (♣). So $\mathcal{K}^\bot$ is unsatisfiable. To show the "only if" direction of *(2)*, assume that $\sigma \in G$ and $\sigma \in \mathsf{cert}(q, \mathcal{K})$. Then there exists a hyperresolution ρ_2 of $P_q(\mathbf{x})\sigma$ from $\mathcal{K} \cup \mathcal{R}_q$. Similarly, by (♣), we know that $\mathsf{support}(\rho_2) \subseteq \mathcal{K}^{q,G}$ and hence $\sigma \in \mathsf{cert}(q, \mathcal{K}^{q,G})$.

We next show inductively a statement from which (♣) will follow. Let $\rho = (\lambda, T)$ be a derivation of a goal in $\Upsilon(q, G)$ from $\mathcal{K} \cup \mathcal{R}_q$, and let $\mathcal{H} = \mathsf{split}(\mathcal{K})$. We have already established (see proof of Theorem 8.11) that $\mathsf{c\text{-}Chase}_\mathcal{K}$ is a model of $\mathcal{H}$. By Lemma 2.5, there exists a constant-preserving homomorphism τ from $\mathsf{skChase}(\mathcal{H})$ to $\mathsf{c\text{-}Chase}_\mathcal{K}$. We show the following properties inductively for every node v in T.

a. $\mathsf{track}(\mathcal{K}, q, G) \models \alpha^R\tau$, for each atom α in $\lambda(v)$; and

b. $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{Rel}(d_r)$, if $\mathsf{sk}(r)$ is the main premise used to obtain $\lambda(u)$ where u is the parent of v.

We proceed by top-down induction on the structure of T.

Base case: In the base case we have that v is the root of T. Property (b) follows vacuously since v has no parent in ρ.

- If $q = \bot$, then ρ is a derivation of the empty clause and $\lambda(v)$ is the empty disjunction and. So property (a) also follows vacuously.
- Otherwise, $\lambda(v) = P_q(\mathbf{x})\sigma$ for some $\sigma \in G$. By the definition of $\mathsf{track}(\mathcal{K}, q, G)$ (point *(iii)*) we have that $(\lambda(v))^R \in \mathsf{track}(\mathcal{K}, q, G)$ and hence property (a) also holds.

Inductive step: Assuming that properties (a) and (b) hold for a node u, we show that they also hold for the children $v_1, \ldots, v_n$ of u. Let r be the rule in $\mathcal{K}$ such that $\mathsf{sk}(r)$ is the main premise in the relevant hyperresolution step with MGU σ, i.e., $\lambda(u) = \gamma_1\sigma \vee \cdots \vee \gamma_m\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is the hyperresolvent of $\mathsf{sk}(r) = \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \gamma_1 \vee \cdots \vee \gamma_m$ and $\lambda(v_i) = \beta_i\sigma \vee \chi_i$ for $1 \leq i \leq n$, using σ.

By Lemma 8.5 in Section 8.1 we have that each $\beta_i\sigma \in \mathsf{skChase}(\mathcal{H})$ for each $1 \leq i \leq n$. Since τ is a homomorphism from $\mathsf{skChase}(\mathcal{H})$ into $\mathsf{c\text{-}Chase}_\mathcal{K}$ we then have that $(\beta_i\sigma)\tau \in \mathsf{c\text{-}Chase}_\mathcal{K}$ and by Proposition 2.4, $\beta_i(\sigma\tau) \in \mathsf{c\text{-}Chase}_\mathcal{K}$ for $1 \leq i \leq n$. We next show that $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{move}(r)\sigma\tau$.

- If $m = 0$, then $\mathsf{move}(r) = P^R_\bot$. We distinguish two cases.
  - if $q \neq \bot$, $P^R_\bot \in \mathsf{track}(\mathcal{K}, q, G)$ by point *(iv)*;
  - if $q = \bot$, we have $\bot_s \in \mathsf{c\text{-}Chase}_\mathcal{K}$ and hence $P^R_q \in \mathsf{track}(\mathcal{K}, q, G)$ by point *(iii)*.
- Otherwise, by the induction hypothesis, we also have that $\mathsf{track}(\mathcal{K}, q, G) \models (\gamma_j\sigma)^R\tau$ and again by Proposition 2.4, $\mathsf{track}(\mathcal{K}, q, G) \models \gamma^R_j(\sigma\tau)$ for $1 \leq j \leq m$.

Therefore $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{move}(r)\sigma\tau$. Then the body of rules in $\Delta(r)$ is satisfied by the substitution $\sigma\tau$ and hence $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{Rel}(d_r)$, and $\mathsf{track}(\mathcal{K}, q, G) \models \beta^R_i(\sigma\tau)$ for $1 \leq i \leq n$. Again by Proposition 2.4, $\mathsf{track}(\mathcal{K}, q, G) \models (\beta^R_i\sigma)\tau$ for $1 \leq i \leq n$. In addition, by the induction hypothesis, we have $\mathsf{track}(\mathcal{K}, q, G) \models \chi^R_i\tau$, for each $1 \leq i \leq n$. Hence, have shown that (a), (b) hold for each child v_i of u.

It only remains to be shown that (a) and (b) imply (♣). Indeed, take any $\alpha \in \mathsf{support}(\rho)$.

- If α is a fact in $\mathcal{K}$, then it is a leaf node of ρ; hence, by property (a) we have that $\mathsf{track}(\mathcal{K}, q, G) \models \alpha^R\tau$. But then, since α is a fact in $\mathcal{D}_\mathcal{K}$ the definition of homomorphism ensures that $\alpha^R\tau = \alpha^R$. By the definition of $\mathcal{K}^{q,G}$ this implies that $\alpha \in \mathcal{K}^{q,G}$.

- If α is a rule in $\mathcal{K}$, then by property (b) we have that $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{Rel}(d_\alpha)$. Again, the definition of $\mathcal{K}^{q,G}$ ensures that $\alpha \in \mathcal{K}^{q,G}$.

This completes the proof of the theorem. □

Comparison with Magic Sets The idea of inverting rules for recording relevant information has been heavily exploited in Logic Programming. In particular, the magic set transformation (Bancilhon et al., 1986) is a technique that, given a program and a query, optimises the materialisation process so as to derive only facts that are relevant to the query. Similarly to our tracking encoding, the magic sets technique uses auxiliary predicates, called "magic" predicates, to identify the relevant facts. This technique was originally developed for datalog, and was subsequently extended to to handle also negation as failure (Beeri et al., 1987; Kemp et al., 1995) and disjunctions (Alviano et al., 2012a).

In contrast to magic sets, the goal of our transformation is not to reduce the size of the materialisation, but rather to compute a relevant fragment of a knowledge base potentially given in a very expressive (even undecidable) language, and to reduce this computation to datalog reasoning. In this sense, our technique is orthogonal to magic sets. Indeed, the benefits of our technique are only relevant for knowledge bases containing existentially quantified and/or disjunctive rules (if $\mathcal{K}$ is datalog, then the query would have been fully answered by the lower bound).

Furthermore, it is worth noticing that the way we invert (datalog) rules is also different from magic sets and yields a more "precise" tracking. This is so because our assumption is that tracking starts with an already computed materialisation (see Point *(i)* in Definition 9.1). For instance, given an already adorned rule $A(x) \wedge B(x) \rightarrow C(x)$, magic sets would produce the following rules for deriving the magic predicates A^M for A and B^M for B:

$$C^M(x) \rightarrow A^M(x) \qquad\qquad C^M(x) \wedge A(x) \rightarrow B^M(x)$$

These rules can be used to derive a fact $A^M(a)$ from $C^M(a)$, even if $A(a)$ cannot be used to derive $C(a)$ because the aforementioned rule is not applicable (e.g., if $B(a)$

does not hold and $C(a)$ is derived using other rules). Our transformation, in contrast, would yield the more restrictive rules

$$C^R(x) \wedge A(x) \wedge B(x) \rightarrow A^R(x) \qquad C^R(x) \wedge A(x) \wedge B(x) \rightarrow B^R(x)$$

which are applicable to a only if both $A(a)$ and $B(a)$ hold in the materialisation.

9.3 Optimisations of the Datalog Encoding

To conclude this section, we present three optimisations of the datalog encoding in Definition 9.1 that we will exploit in our system PAGOdA.

9.3.1 Subsumption in Hyperresolution

The **first** optimisation aims at reducing the size of the computed subsets. Recall that the key step in the construction of the tracking knowledge base $\mathsf{track}(\mathcal{K}, q, G)$ was to invert the rules in $\mathcal{K}$ to capture hyperresolution proofs in a "backwards" fashion. Consider the inversion (9.4) of rule (R5) in our running example. The effect of the inversion is to capture the applicability of hyperresolution: if facts $\mathsf{Mammal}(\mathit{rabbit})$, $\mathsf{Herbivore}^R(\mathit{rabbit})$ and $\mathsf{MeatEater}^R(\mathit{rabbit})$ hold, then we include rule (R5) in the subset since there may be a proof in $\mathcal{K}$ involving a step where a ground clause $\mathsf{Herbivore}(\mathit{rabbit}) \vee \mathsf{MeatEater}(\mathit{rabbit}) \vee \xi$ is obtained by resolving (R5) with $\mathsf{Mammal}(\mathit{rabbit}) \vee \xi$.

Note, however, that such a step is redundant should $\mathsf{Herbivore}(\mathit{rabbit})$ already be contained in $\mathcal{K}$, in which case (R5) may not be needed in the relevant subset. We can capture this observation by distinguishing in the tracking knowledge base those facts in the c-chase of $\mathcal{K}$ that were not already present in the original dataset $\mathcal{D}_\mathcal{K}$. We encode these "implied" facts by instantiating fresh predicates P^I for each predicate P in $\mathcal{K}$. In our running example, a fact $\mathsf{MeatEater}^I(\mathit{rabbit})$ in the tracking knowledge base establishes that $\mathsf{MeatEater}(\mathit{rabbit})$ was not present in the original data. We then use atoms over these predicates as guards in the inverted rules, e.g. rule (9.4) would now be written as follows:

$$\mathsf{Herbivore}^I(x) \wedge \mathsf{MeatEater}^I(x) \wedge \mathsf{Herbivore}^R(x) \wedge \mathsf{MeatEater}^R(x) \wedge \mathsf{Mammal}(x) \rightarrow \mathsf{Mammal}^R(x)$$

Formally, Definition 9.1 can be optimised as given next.

Definition 9.5. Let $\mathcal{K}$, q, G and predicates P^R be as in Definition 9.1. For each predicate P, let P^I be a fresh predicate of the same arity as P. We now redefine $\mathsf{move}(r)$ for each rule r as the following conjunction of atoms:

- $P^R_\bot$ if r of the form ($\bot$-`Norm`);
- $\gamma^I_1(\mathbf{x}, \mathbf{z}_1) \wedge \gamma^R_1(\mathbf{x}, \mathbf{z}_1)$ if r of the form ($\exists$-`Norm`); and
- $\gamma^I_1(\mathbf{x}) \wedge \cdots \wedge \gamma^I_m(\mathbf{x}) \wedge \gamma^R_1(\mathbf{x}) \wedge \cdots \wedge \gamma^R_m(\mathbf{x})$ if r of the form ($\vee$-`Norm`).

Then, $\Delta(r)$ is as in Definition 9.1, and $\mathsf{track}(\mathcal{K}, q, G)$ is also as in Definition 9.1, but extended with the addition of a fact $P^I(\mathbf{x})\sigma$ for each fact $P(\mathbf{x})\sigma$ that is in $\mathsf{c\text{-}Chase}_\mathcal{K}$ but not in $\mathcal{D}_\mathcal{K}$. ♢

It is easy to see that this optimisation does not affect the correctness of Theorem 9.4: if a disjunction of atoms is derived via hyperresolution, where one of the atoms is already present in the data, then the disjunction is subsumed and can be dispensed with.

Furthermore, we could also start with facts derived in the datalog materialisation M^L_1 (see Step 1 in Section 7.3) rather than $\mathcal{D}_\mathcal{K}$. The reason behind is that we know all facts in M^L_1 are sound, and so there is no need to trace back the proof of these facts. In this case, the extracted dataset is a subset of M^L_1 rather than $\mathcal{D}_\mathcal{K}$. For simplicity in presentation, we use $\mathcal{D}_\mathcal{K}$ through out the theoretic work. In the experiment, we tried both and finally chose to use M^L_1 as the start point.

9.3.2 Native Equality Reasoning

The **second** optimisation can be used to obtain a more succinct encoding for datalog reasoners that support equality reasoning natively (such as RDFox). As already mentioned, the built-in semantics of the equality predicate can be axiomatised within datalog. However, axiomatisation can lead to performance issues, and scalability can be improved by a native treatment of equality where equal objects are "merged" into a single representative of the whole equivalence class.

The axiomatisation of equality has a significant effect in our tracking encoding. For example, the replacement rules r of the form ($\mathtt{REP_P}$) are inverted into the following rules in $\Delta(r)$ for each predicate P:

$$P^R(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) \wedge P(x_1, \ldots, x_n) \wedge x_i \approx y \rightarrow P^R(x_1, \ldots, x_n) \quad (9.9a)$$

$$P^R(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) \wedge P(x_1, \ldots, x_n) \wedge x_i \approx y \rightarrow \approx^R (x_i, y) \quad (9.9b)$$

where (9.9a) is an tautology and can be dispensed with, but rule (9.9b) is required. If the datalog reasoner has native support for equality, then we do not need to include in the tracking knowledge base the inversion of equality axioms (`REF`), (`SYM`) or (`TRA`), and we only need to include rules (9.9b) in order to ensure that the computed subset has the required properties. The result is a more succinct encoding that can be materialised more efficiently.

Example 9.6. Consider a knowledge base $\mathcal{K}$ consists of facts $\{R(a_1, b), R(a_2, b), A(a_1)\}$ and the following rules.

$$A(x) \to B(x) \vee C(x) \qquad B(x) \to D(x)$$
$$R(x_1, y) \wedge R(x_2, y) \to x_1 \approx x_2 \qquad C(x) \to D(x)$$

Let $q = D(x)$, the gap G between lower and upper bounds to q is $\{a_1, a_2\}$. It is easy the see that rule $R(x_1, y) \wedge R(x_2, y) \to x_1 \approx x_2$ is essential to derive $q(a_2)$. To ensure that this rule is in the fragment $\mathcal{K}^{q,G}$, we have to track $a_1 \approx a_2$ using an instance of rule (9.9b). ♢

9.3.3 A $\bot$-Related Optimisation

The **third** optimisation is a more fine-grained treatment for $\bot$. As briefly discussed after Definition 9.1, $\mathcal{K}^{\bot}$ of $\mathcal{K}$ is a subset of the relevant subset of an arbitrary query q', i.e. $\mathcal{K}^{\bot} \subseteq \mathcal{K}^{q',G}$ for any set of possible answers G to q'. Sometimes, $\mathcal{K}^{\bot}$ should be included into relevant subsets of queries due to the interaction between rules of the form ($\bot$-`Norm`) and rules of the form ($\vee$-`Norm`).

Example 9.7. Consider a query $q(x) = Q(x)$ and a knowledge base consisting of the following rules and facts.

$$A(x) \to B(x) \vee C(x) \vee D(x) \qquad C(x) \to Q(x) \qquad D(x) \to Q(x)$$
$$R(x, y) \wedge B(x) \to \bot \qquad A(a) \qquad R(a, b)$$

Then $\mathcal{K}^{\bot} = \{R(x, y) \wedge C(x) \to \bot, R(a, b)\}$. In this case $\mathcal{K}^{\bot}$ is necessary to derive the certain answer a as the rule $R(x, y) \wedge B(x) \to \bot$ and the fact $R(a, b)$ together rule out the possibility to derive $B(a)$ for the first rule on $\{x \mapsto a\}$. ♢

It is, however, suboptimal to include $\mathcal{K}^{\bot}$ into relevant subsets for queries all the time since there can be queries that has no interaction with $\bot$, which is illustrated the following simple example.

Example 9.8. Consider a query $q(x) = Q(x)$ and a knowledge base consisting of the following rules and facts.

$$\begin{array}{lll} A(x) \to B(x) \vee D(x) & C(x) \to E(x) \vee F(x) & \\ B(x) \to Q(x) & E(x) \wedge F(x) \to \bot & \\ D(x) \to Q(x) & A(a) & C(c) \end{array}$$

Then $\mathcal{K}^{\bot} = \{C(x) \to D(x) \vee E(x), D(x) \wedge E(x) \to \bot, C(c)\}$. It is not difficult to tell that none of rules or facts in $\mathcal{K}^{\bot}$ participates in a hyperresolution derivation of $q(a)$ since the predicate Q occurring in q doesn't depend on D or E. That means $\mathcal{K}^{\bot}$ is irrelevant to verify answers to q. $\diamondsuit$

Inspired by the above examples, we have developed techniques to compute a finite set $\{\mathcal{K}^{\bot_1}, \mathcal{K}^{\bot_2}, \dots\}$ of subsets of $\mathcal{K}^{\bot}$ and then to decide which subsets are relevant. Intuitively, we can treat each occurrence of $\bot$ as a different nullary predicate $\bot_i$ and compute relevant subsets $\mathcal{K}^{\bot_i}$ for each $\bot_i$.

Example 9.9. Consider a query $q(x) = Q(x)$, and a knowledge base $\mathcal{K}$ consisting of the following rules and facts.

$$\begin{array}{lll} A(x) \to B(x) \vee D(x) \vee F(x) & C(x) \to D(x) \vee E(x) & A(a) \\ R(x,y) \wedge D(x) \to \bot & D(x) \wedge E(x) \to \bot & R(a,b) \\ B(x) \to Q(x) & F(x) \to Q(x) & C(c) \end{array}$$

Then $\mathcal{K}^{\bot} = \mathcal{K} \setminus \{B(x) \to Q(x), F(x) \to Q(x)\}$, and the computed relevant subset for q and $\{a\}$ under Definition 9.1 is $\mathcal{K}$ itself. In order to treat each occurrence of $\bot$ separately, we first assign a positive number $N_{\bot}(r)$ for each rule r of the form ($\bot$-`Norm`). Let $N_{\bot}(R(x,y) \wedge D(x) \to \bot) = 1$ and $N_{\bot}(D(x) \wedge E(x) \to \bot) = 2$. Then $\mathcal{K}^{\bot}$ can be further separated into two parts as follows.

$$\begin{array}{l} \mathcal{K}^{\bot_1} = \{A(x) \to B(x) \vee D(x) \vee F(x),\ R(x,y) \wedge D(x) \to \bot,\ A(a),\ R(a,b)\}, \\ \mathcal{K}^{\bot_2} = \{C(x) \to D(x) \vee E(x),\ D(x) \wedge E(x) \to \bot,\ C(c)\}. \end{array}$$

We can observe that $\mathcal{K}^{\bot_1}$ is relevant to q and $\{a\}$ as it blocks the application of $A(x) \to B(x) \vee D(x) \vee F(x)$ on $\{x \mapsto a\}$ deriving $D(a)$. Moreover, $\mathcal{K}^{\bot_2}$ is not relevant to q and $\{a\}$ since it does not lead to any facts about constant a. $\diamondsuit$

The definition of tracking knowledge base with this optimisation, as well as the correctness proof, is technically involved. So we defer them to Appendix A. This

optimisation allows to exclude some rules and facts in the relevant subset of $\bot$ from relevant subsets of queries, and hence we can compute smaller subsets for the later verification of gap answers.

Example 9.10. By adopting this optimisation on $\bot$-related subsets, we can get the $\mathcal{K}^{\bot_1}$ and $\mathcal{K}^{\bot_2}$ in Example 9.9. Finally, the computed relevant subset of q and $\{a\}$ is $\mathcal{K} \setminus \mathcal{K}^{\bot_2}$, which is a strict subset of $\mathcal{K}$—the computed relevant subset under Definition 9.1.

Chapter 10

Fully-fledged Reasoning

In this chapter, let q be an input query different from the unsatisfiability query $\bot$. Once $\mathcal{K}^{\bot}$ and $\mathcal{K}^{q}$ have been computed, we still need to check, using the fully-fledged reasoner, the satisfiability of $\mathcal{K}^{\bot}$ as well as whether $\mathcal{K}^{q}$ entails each candidate answer in G^{q}. This can be computationally expensive if these subsets are large and complex, or there are many candidate answers to verify. So two additional optimisations have been deployed to before calling a fully-fledged reasoner (see Section 10.1). Moreover, the capability of existing fully-fledged OWL 2 reasoners to answer conjunctive queries are restricted to queries of a certain shape, such as, ground queries or tree-shaped queries. In Section 10.2, we introduce a rolling-up technique that allows to answer a more general class of conjunctive queries.

10.1 Optimisations

The first optimisation aims at filtering out some spurious answers by merge constants that are "similar" to each other. The resulted knowledge base is called a summary which becomes logically stronger. The purpose of the second optimisation is to reduce the number of calls to the fully-fledged reasoner by extracting endomorphism relations from possible answers in the gap.

10.1.1 Summarisation

We exploit *summarisation* techniques (Dolby et al., 2007) in an effort to further reduce the number of candidate answers. The idea behind summarisation is to "shrink" the data in the knowledge base by merging all constants that instantiate the same set of unary predicates. Since summarisation is equivalent to extending the knowledge base with equality assertions between constants, the summary knowledge base entails the

original one by the monotonicity of first-order logic. Consequently, we can exploit summarisation as follows:

1. If the satisfiability of $\mathcal{K}$ remains undetermined, we construct the summary of $\mathcal{K}^{\bot}$ and check its satisfiability. If it is satisfiable, then $\mathcal{K}^{\bot}$ (and thus also $\mathcal{K}$) is also satisfiable.

2. Construct the summary of $\mathcal{K}^{q}$ and then use the fully-fledged reasoner to check whether if the summary of a gap answer is entailed to be a certain answer of the corresponding summary of q in the summary of $\mathcal{K}^{q}$, discarding any answers that are not so entailed.

Formally, summarisation is defined as follows.

Definition 10.1. A *type* T is a set of unary predicates; given a constant c in $\mathcal{K}$, we say that $T = \{A \mid A(c) \in \mathcal{K}\}$ is the type for c. For each type T, let a_T be a fresh constant uniquely associated with T. The summary function over $\mathcal{K}$ is the substitution θ mapping each constant c in $\mathcal{K}$ to a_T, where T is the type for c. Finally, the summary of $\mathcal{K}$ is $\mathcal{K}\theta$. ♢

The following proposition shows how summarisation can be exploited to detect spurious answers in our setting. Since summarisation can significantly reduce data size in practice, and the relevant subsets $\mathcal{K}^{\bot}$ and $\mathcal{K}^{q}$ are already significantly smaller than $\mathcal{K}$, checking the satisfiability of $\mathcal{K}^{\bot}$ and of each gap answer against $\mathcal{K}^{q}$ becomes feasible in many cases, even though doing so implies resorting to the fully-fledged reasoner.

Proposition 10.2. *Let θ be the summary function over $\mathcal{K}$. We have $\mathsf{cert}(q, \mathcal{K})\theta \subseteq \mathsf{cert}(q\theta, \mathcal{K}\theta)$ for every conjunctive query q.*

Proof. Let $G(\mathcal{K})$ be the knowledge base

$$G(\mathcal{K}) = \mathcal{K} \cup \{c \approx a_T \mid T = \{A \mid A(c) \in \mathcal{K}\}\}.$$

Then we have $G(\mathcal{K})$ is a conservative extension of $\mathcal{K}\theta$ and $G(\mathcal{K})\theta = \mathcal{K}\theta$. Due to the monotonicity of first-order logic, we have $G(\mathcal{K}) \models \mathcal{K}$. So we have $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, G(\mathcal{K}))$ and hence $\mathsf{cert}(q, \mathcal{K})\theta \subseteq \mathsf{cert}(q, G(\mathcal{K}))\theta$. It is easy to verify from both directions that $\mathsf{cert}(q, G(\mathcal{K}))\theta = \mathsf{cert}(q\theta, \mathcal{K}\theta)$. Thus, $\mathsf{cert}(q, \mathcal{K})\theta \subseteq \mathsf{cert}(q\theta, \mathcal{K}\theta)$. □

Example 10.3. In the case of our running example, the remaining gap answers are *rabbit*, *tiger*, and *lion*. Constants *tiger* and *lion* both have type $\{\mathsf{Mammal}\}$, and are therefore mapped to a fresh constant, say t_{Mammal}, that is uniquely associated with type $\{\mathsf{Mammal}\}$ whereas the constant *rabbit* is mapped into the constant $t_{\mathsf{Herbivore}}$. Since t_{Mammal} is not a certain answer to q_{ex} w.r.t. the summary of $\mathcal{K}_{ex}$, we can determine that both *tiger* and *lion* are spurious answers. Now it remains only a single answer *rabbit* to be verified. ♢

10.1.2 Analysis of Answer Dependencies

If summarisation did not suceed in pruning all candidate answers in G, we try in a last step to further reduce the calls to the fully-fledged reasoner by exploiting dependencies between the remaining candidate answers such that, if answer σ depends on answer π, and σ is spurious, then so is π.

Consider two possible answers π and σ in G^q. Suppose that we can find an endomorphism η from $\mathbf{x}\pi$ to $\mathbf{x}\sigma$ in the dataset $\mathcal{D}_{\mathcal{K}}$. If we can determine (by calling the fully-fledged reasoner) that σ is a spurious answer, then so must be π; as a result, we no longer need to call the fully-fledged reasoner to verify π. Such endomorphisms are defined next.

Definition 10.4. $\mathbf{c} = (c_1, \ldots, c_n)$ and $\mathbf{d} = (d_1, \ldots, d_n)$ be n-tuples of constants from $\mathcal{K}$. An *endomorphism* from $\mathbf{c}$ to $\mathbf{d}$ in $\mathcal{K}$ is a mapping η from constants to constants such that *(i)* $c_i\eta = d_i$ for each $1 \leq i \leq n$; *(ii)* $P(t_1, \ldots, t_m)\eta \in \mathcal{D}_{\mathcal{K}}$ for each fact $P(t_1, \ldots, t_m) \in \mathcal{D}_{\mathcal{K}}$; and *(iii)* $r\eta \in \Sigma_{\mathcal{K}}$ for each $r \in \Sigma_{\mathcal{K}}$. ♢

The relevant property of endomorphisms is given in the following proposition.

Proposition 10.5. *Let π, σ be two possible answers to q and let η be an endomorphism from $\mathbf{x}\pi$ to $\mathbf{x}\sigma$ in $\mathcal{K}$. Then, $\pi \in \mathsf{cert}(q, \mathcal{K})$ implies $\sigma \in \mathsf{cert}(q, \mathcal{K})$.*

Proof. Since $\pi \in \mathsf{cert}(q, \mathcal{K})$, we know that $\mathcal{K} \models q(\mathbf{x})\pi$. So there is a hyperresolution derivation $\rho = (T, \lambda)$ of $P_q(\mathbf{x})\pi$ from $\mathcal{K} \cup \mathcal{R}_q$. It is easy to check that $(T, \lambda \circ \eta)$ is a hyperresolution derivation of $P_q(\mathbf{x})\sigma$ from $\mathcal{K} \cup \mathcal{R}_q$. Then, $\mathcal{K} \models q(\mathbf{x})\sigma$ and hence $\sigma \in \mathsf{cert}(q, \mathcal{K})$. □

Example 10.6. Consider a knowledge base consisting of rules and facts $A(x) \rightarrow C(x), A(a), A(b), B(b)$. There is an endomorphism from a to b in the dataset. In this case, for query $q(x) = C(x)$, if we find that $x \mapsto a$ is a certain answer to q, we know that $x \mapsto b$ is also a certain answer without resorting to a fully-fledged reasoner. ♢

We exploit this idea to compute a dependency graph having candidate answer tuples as nodes and an edge (π, σ) whenever an endomorphism in $\mathcal{D}_{\mathcal{K}}$ exists mapping $\mathbf{x}\pi$ to $\mathbf{x}\sigma$. Computing endomorphisms is, however, a computationally hard and we will resort in practice to a sound greedy algorithm to compute a partial dependency graph.

10.2 Query Internalisation

As described in the beginning of this chapter, our approach relies on a fully-fledged reasoner to check conjunctive query entailment. In general, conjunctive query entailment is challenging and computationally expensive. Decidability for knowledge bases stemming from OWL 2 remains open. Existing OWL 2 reasoners, however, are typically capable to answer conjunctive queries that are ground or tree-shaped. In this chapter, we show how to use an existing OWL 2 reasoner to answer the more general class of internalisable queries defined in Definition 4.10.

In Section 10.2.1, we introduce the notion of query graph, which will be exploited to formalise our rolling-up technique in Section 10.2.2. Later, in Section 10.2.3, we exploit our rolling-up technique to show that answering internalisable queries can be reduced to entailment of data assertions and ontology satisfiability.

10.2.1 Query Graphs

Before defining query graphs, we introduce a mixed notation of DL and first-order logic that will be exploited in query graphs.

Definition 10.7. *Query concepts* are formed according the following syntax:

$$C \longrightarrow \{c\} \mid \{x\} \mid A \mid \exists R.\mathsf{Self} \mid \exists R.C \mid C_1 \sqcap C_2$$

where A stands for an atomic concept, c a constant, x a variable, and R a role. A *query atom* is of the form $C(t)$ for a query concept C or $P(t_1, t_2)$ for an atomic role P. A query atom is *ground* if it contains no variables.

The application of a substitution can be extended to query concepts naturally. The result of applying σ to a query concept C is defined as follows:

$$C\sigma = \begin{cases} \{\sigma(x)\} & \text{if } C \text{ is of the form } \{x\} \text{ with variable } x, \\ \exists R.(C'\sigma) & \text{if } C \text{ is of the form } \exists R.C', \\ (C_1\sigma) \sqcap (C_2\sigma) & \text{if } C \text{ is of the form } C_1 \sqcap C_2, \\ C & \text{otherwise.} \end{cases}$$

The application of σ to query atoms $C(t)\sigma$ and $P(t_1, t_2)\sigma$ are defined as query atoms $(C\sigma)(t\sigma)$ and $P(t_1\sigma, t_2\sigma)$, respectively. $\diamondsuit$

Now we are ready to define query graphs.

Definition 10.8. A *query graph* $G = \langle V_f, V_e, V_c, E, \rho \rangle$ is a labeled directed multi-graph where

- V_f, V_e, V_c are pair-wise disjoint sets of free variables, existentially quantified variables and constants, respectively,
- ρ maps each term u in $V_f \cup V_e \cup V_c$ to a set $\rho(u)$ of query concepts, and
- E is a set of directed edges $u \xrightarrow{P} v$ from u to v labeled with an atomic role P.

A query graph is *atomic* if $\rho(u)$ is a set of atomic concepts for each $u \in V$.

Each edge $u \xrightarrow{P} v \in E$ is associated with two *sides*, written as $u \xrightarrow{P} v$ and $v \xrightarrow{P^-} u$. We write for a side $u \xrightarrow{R} v \bar{\in} G$ if an edge in E is associated with this side.

An *e-cycle* in G is defined as a sequence $u_0 \xrightarrow{R_1} u_1 \xrightarrow{R_2} \dots \xrightarrow{R_n} u_n$ such that $n \geq 2$, $u_0 = u_n$, $u_i \in V_e$ and $u_{i-1} \xrightarrow{R_i} u_i \bar{\in} G$ for each $1 \leq i \leq n$, and $R_{i-1} \neq \mathsf{inv}(R_i)$ or $u_{i-2} \neq u_i$ for each $2 \leq i \leq n$. The set of all e-cycles in G is denoted by $\mathsf{e\text{-}cyc}(G)$. The query graph G is *e-acyclic* if $\mathsf{e\text{-}cyc}(G) = \emptyset$. A term in $V_f \cup V_e \cup V_c$ is *isolated* in G if it has no incoming edges or outgoing edges. $\diamondsuit$

Semantics of Query Graphs Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation, let $G = \langle V_f, V_e, V_c, E, \rho \rangle$ be a query graph, and let σ be substitution from V_f to $\Delta^{\mathcal{I}}$. A *witness* of G and σ in $\mathcal{I}$ is a substitution σ' from V_e to $\Delta^{\mathcal{I}}$ such that

- $\mathcal{I} \models (P(u, v))\sigma\sigma'$ for each $u \xrightarrow{P} v \in E$, and
- $\mathcal{I} \models (C(u))\sigma\sigma'$ for each $u \in V$ and $C \in \rho(u)$.

The interpretation $\mathcal{I}$ satisfies G and σ if there exists a witness of G and σ in $\mathcal{I}$. By slight abuse of notation, we write $\mathcal{I} \models G\sigma$ to denote that $\mathcal{I}$ satisfies G and σ and call such a substitution σ an *answer* to G in $\mathcal{I}$. Since all query atoms $P(u, v)\sigma\sigma'$ and $C(u)\sigma\sigma'$ are ground, the semantics of $\mathcal{I} \models P(u, v)\sigma\sigma'$ and $\mathcal{I} \models (C(u))\sigma\sigma'$ is defined by the semantics of DLs.

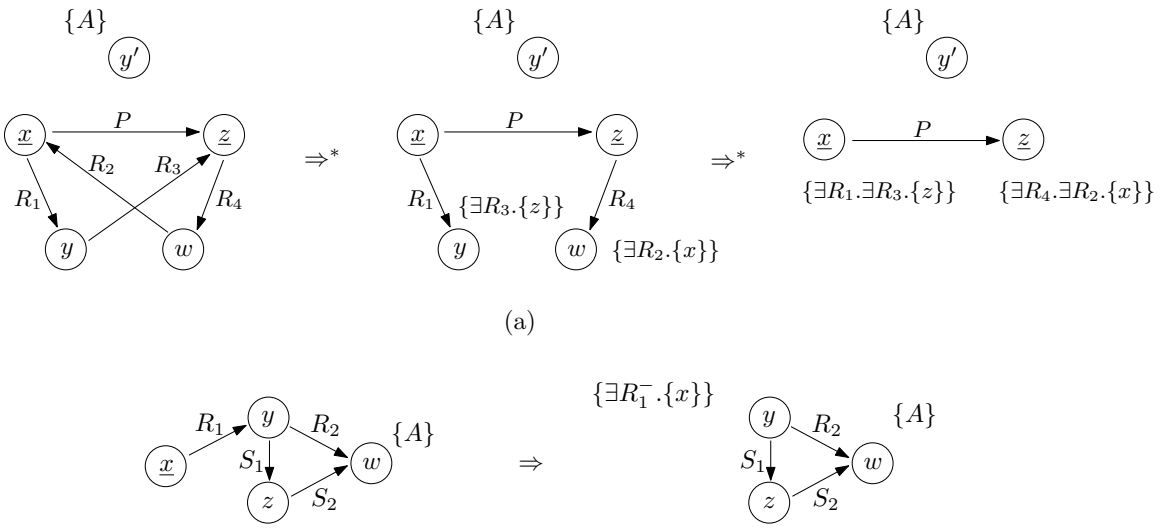


Figure 10.1: Internalisation of queries

Canonical Graph of Queries Each conjunctive query can be seen as an atomic query graph defined as follows.

Definition 10.9. Let q be a conjunctive query. The *canonical graph* $\mathcal{G}(q)$ is the following atomic query graph $\langle V_f, V_e, V_c, E, \rho \rangle$:

- $V_f = \mathsf{f\text{-}var}(q)$, $V_e = \mathsf{e\text{-}var}(q)$, $V_c = \mathsf{const}(q)$,
- $\rho(u) = \{A \mid A(u) \in q\}$ for each term $u \in V$, and
- $E = \{u \xrightarrow{P} y \mid P(u, v) \in q\}$. ♢

In contrast to term graphs as in Definition 4.4, edges in canonical graphs are directed and each term is labeled with a set of query concepts. Despite these differences, we can alternatively characterise internalisable queries by their canonical graphs by the following proposition.

Proposition 10.10. *A conjunctive query q is internalisable iff $\mathcal{G}(q)$ is e-acyclic.*

The correctness of the proposition is trivial as the term graph $\mathcal{U}(q)$ of a query q can be obtained from $\mathcal{G}(q)$ by ignoring edge directions and term labels. Then, by the definition of e-cycle in Definition 10.8, we have that an e-cycle in $\mathcal{G}(q)$ corresponds to a cycle of length at least two involving only existentially quantified variables in $\mathcal{U}(q)$.

Example 10.11. Figure 10.1 presents the canonical graphs of queries (c) and (d) in Figure 4.2, where free variables are underlined. The cycle y, z, w in Figure 4.2d corresponds to the e-cycle $y \xrightarrow{S_1} z \xrightarrow{S_2} w \xrightarrow{R_2^-} y$ in Figure 10.1b. ♢

A key property of the canonical graph of a conjunctive query is that it preserves entailment of query answers w.r.t. an interpretation.

Proposition 10.12. *Let q be a conjunctive query, let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation, and let σ be a substitution from $\mathsf{f\text{-}var}(q)$ to $\Delta^{\mathcal{I}}$. Then*

$$\mathcal{I} \models q\sigma \text{ iff } \mathcal{I} \models \mathcal{G}(q)\sigma.$$

Since $\mathcal{G}(q)$ is atomic, the semantics of $\mathcal{I} \models \mathcal{G}(q)\sigma$ coincides with the semantics of $\mathcal{I} \models q\sigma$ and hence the proposition holds.

10.2.2 Rolling-up

Our rolling-up technique is an extension of the rolling-up for $\mathcal{ALC}$ (Horrocks and Tessaris, 2000). We first define our rolling-up operation on query graphs and then show that the operation preserves answers w.r.t. an interpretation.

Definition 10.13. Let $G = \langle V_f, V_e, V_c, E, \rho \rangle$ be a query graph. A term $y \in V_y$ is a *leaf* if it is connected to a single incoming edge or a single outgoing edge. A side $u \xrightarrow{R} v \in G$ is *applicable* if v is a leaf or $u \in V_e$ and $v \in \{u\} \cup V_f \cup V_c$. Moreover, a query graph is *complete* if it contains no applicable sides.

The *rolling-up* operation $\Omega(\cdot, \cdot)$ is defined on a query graph and an applicable side in the graph. Let $u \xrightarrow{R} v$ be an applicable side in G, and let $e \in E$ be the edge associated with this side. Then $\Omega(G, u \xrightarrow{R} v)$ is defined as the following query graph $G' = \langle V_f', V_e', V_c', E', \rho' \rangle$:

- $V_f' = V_f$, $V_c' = V_c$, $V_e' = V_e \setminus \{v\}$,
- $E' = E \setminus \{e\}$, and
- ρ' is given as follows for each term w in $V_f' \cup V_e' \cup V_c'$.

$$\rho'(w) = \begin{cases} \rho(w) & \text{if } w \neq u, \\ \rho(w) \cup \{\exists R.(\sqcap_{D \in \rho(v)} D)\} & \text{if } w = u \text{ and } v \in V_e, \\ \rho(w) \cup \{\exists R.\mathsf{Self}\} & \text{if } w = u \text{ and } u = v, \\ \rho(w) \cup \{\exists R.\{v\}\} & \text{if } w = u \text{ and } v \in V_f \cup V_c. \end{cases} \quad \diamondsuit$$

Clearly, the range of ρ' is a set of query concepts since $\exists R.(\bigsqcap_D D)$, $\exists R.\mathsf{Self}$ and $\exists R.\{v\}$ for each $v \in V_f \cup V_c$ are all query concepts. The rolling-up operation is answer-preserving over query graphs, which can be formulated as follows.

Lemma 10.14. *Let $G = \langle V_f, V_e, V_c, E, \rho \rangle$ be a query graph, let $u \xrightarrow{R} v$ be an applicable side in G, let $G' = \Omega(G, u \xrightarrow{R} v)$, let $\mathcal{I}$ be an arbitrary interpretation, and let σ be a substitution from V_f to $\Delta^{\mathcal{I}}$. We have*

$$\mathcal{I} \models G\sigma \text{ iff } \mathcal{I} \models G'\sigma.$$

Proof. Let $G' = \langle V_f', V_e', V_c', E, \rho' \rangle$. To prove the "only if" direction, we assume that $\mathcal{I}$ is a model of $G\sigma$. Let σ' be a witness of G and σ. It suffices to show that $\mathcal{I} \models C(u)\sigma\sigma'$ for the concept C added to $\rho'(u)$ in this rolling-up step. We distinguish three cases:

- if $u = v$, then $C = \exists R.\mathsf{Self}$. Then $\mathcal{I} \models \mathsf{ar}(R, u, u)\sigma\sigma'$ since there is a self-loop in G labeled with the atomic role for R.
- if $v \in V_e$, then $C = \exists R.(\bigsqcap_{D \in \rho(v)} D)$. Then $\mathcal{I} \models \mathsf{ar}(R, u, v)\sigma\sigma'$ and $\mathcal{I} \models D(v)\sigma\sigma'$ for each $D \in \rho(v)$. Thus, $\mathcal{I} \models (\exists R.(\bigsqcap_{D \in \rho(v)} D))(u)\sigma\sigma'$
- if $v \in V_f \cup V_c$, then $C = \exists R.\{v\}$. Then $\mathcal{I} \models \mathsf{ar}(R, u, v)\sigma\sigma'$ and hence $\mathcal{I} \models (\exists R.\{v\})\sigma\sigma'$.

So the projection σ' to V_e' is a witness of G' and σ in $\mathcal{I}$ and hence $\mathcal{I} \models G'\sigma$.

To prove the "if" direction, we assume that $\mathcal{I}$ is a model of $G'\sigma$. Let σ'' be a witness of G' and σ in $\mathcal{I}$. We distinguish three cases:

- if $u = v$, then $\mathcal{I} \models \exists R.\mathsf{Self}(u)\sigma\sigma''$. Then $\mathcal{I} \models \mathsf{ar}(R, u, u)\sigma\sigma'$ for the self-loop in G labeled with the atomic role for R.
- if $v \in V_e$, then $\mathcal{I} \models \exists R.(\bigsqcap_{D \in \rho(v)} D)\sigma\sigma''$. Then there exists a constant c_v in $\Delta^{\mathcal{I}}$ such that $\mathcal{I} \models \mathsf{ar}(R, u, c_v)\sigma\sigma''$ and $\mathcal{I} \models D(c_v)\sigma\sigma'$ for each $D \in \rho(v)$. Extend σ'' to map v to c_y. Thus, $\mathcal{I} \models (\exists R.(\bigsqcap_{D \in \rho(v)} D))(u)\sigma\sigma''$.
- if $v \in V_f \cup V_c$, then $\mathcal{I} \models \exists R.\{v\}\sigma\sigma''$. Then $\mathcal{I} \models \mathsf{ar}(R, u, v)\sigma\sigma''$ and hence $\mathcal{I} \models (\exists R.\{v\})\sigma\sigma''$.

So the (extended) σ'' is a witness of G and σ in $\mathcal{I}$ and hence $\mathcal{I} \models G\sigma$. □

An easy observation for the rolling-up operation is that it also preserves e-cycles in a query graph G, since all sides involved in an e-cycle are not applicable in G.

Proposition 10.15. *Let G be a query graph, and let $u \xrightarrow{R} v$ be an applicable side in G. Then, $\mathsf{e\text{-}cyc}(G) = \mathsf{e\text{-}cyc}(\Omega(G, u \xrightarrow{R} v))$.*

10.2.3 Internalisation

The internalisation process of a conjunctive query q amounts to exhaustively applying the rolling-up operation to the canonical graph of q.

Definition 10.16. Let q be a conjunctive query. Let $\prec$ be an arbitrary but fixed total ordering over all sides in $\mathcal{G}(q)$. The internalisation sequence of q w.r.t. $\prec$ is the sequence of query graphs $G^0, G^1, \ldots, G^n$ such that

- $G^0 = \mathcal{G}(q)$, G^n is complete, and
- for each $1 \leq i \leq n$, $G_i = \Omega(G_{i-1}, u \xrightarrow{R} v)$, where $u \xrightarrow{R} v$ is the minimum applicable side of G^{i-1} w.r.t. $\prec$.

Finally, G^n is called the rolled-up graph of q. ♢

In each step of internalisation, an edge is removed from the query graph and a concept is added to the label of a term. So the internalisation process can be done in linear time w.r.t. the number of atoms in an input query. Moreover, the total size of the labels on $V_f \cup V_e \cup V_c$ is bounded by the number of total number of the labels in the canonical graph, and hence it is linear to the size of the input query.

Let G be a rolled-up graph of a query q. Then G is complete and it contains no applicable sides. So occurrences of existentially quantified variables in G are restricted. If q is internalisable, occurrences of existentially quantified variables in G are further restricted.

Theorem 10.17. *Let $G = \langle V_f, V_e, V_c, E, \rho \rangle$ be the rolled-up graph of a conjunctive query q. Then, q is internalisable iff each $y \in V_e$ is isolated.*

Proof. To show the "if" direction by contraposition, we assume that q is not internalisable. Then $\mathcal{G}(q)$ contains an e-cycle by Proposition 10.10. By Proposition 10.15, we have that this e-cycle is preserved in G. So all variables involved in this e-cycle are not isolated. We prove "only if" direction by contraposition as well. Assuming that a non-isolated $y \in V_e$ exists, we next prove by induction on the index i that there exists an infinite sequence $y_0 \xrightarrow{R_1} y_1 \xrightarrow{R_2} \ldots \xrightarrow{R_n} y_n$ such that each $y_i \in V_e$, $y_{i-1} \xrightarrow{R_i} y_i \bar{\in} G$ for each $1 \leq i \leq n$ and $R_{i-1} \neq \mathsf{inv}(R_i)$ or $y_{i-2} \neq y_i$ for each $2 \leq i \leq n$.

Base case: $[i \leq 1]$ The existence of a side y_0 is guaranteed by our assumption that V_e contains a non-isolated term. Let y_0 be such a non-isolated term in V_e and let y_1 be a term connected to y_0 in G. Then R_1 exists that $y_0 \xrightarrow{R_1} y_1$. As G contains no applicable sides, we have $y_1 \in V_e$.

Inductive step: $[i \geq 2]$ The induction hypothesis is that the side $y_{i-2} \overset{R_{i-1}}{\rightharpoonup} y_i$ exists. Since G contains no applicable sides, G has no leaf either. So there are at least two edges connected to y_{i-1} in G, and y_{i-1} has at least another edge other than the one corresponds to $y_{i-2} \overset{R_{i-1}}{\rightharpoonup} y_i$. Let y_i be the destination of the other edge. Then R_i exists such that $y_{i-1} \overset{R_i}{\rightharpoonup} y_i \bar{\in} G$ and $\mathsf{inv}(R_i) \neq R_{i-1}$ or $y_i \neq y_{i-2}$. Since $y_{i-1} \overset{R_i}{\rightharpoonup} y_i$ is not applicable, we have $y_i \in V_e$.

Since V_e is finite, there must exists a pair $i < j$ such that $y_i = y_j$ in the sequence. Let i, j such a pair that j is minimum. Then $y_i \overset{R_{i+1}}{\rightharpoonup} y_{i+1} \overset{R_{i+2}}{\rightharpoonup} \dots \overset{R_j}{\rightharpoonup} y_j$ is an e-cycle in G. By Proposition 10.15 and Proposition 10.10, we have G is not internalisable. □

Example 10.18. For each example query q in Figure 10.1, we use the following order $\prec_L$ for the internalisation of q: for any terms u, v in q, $u \prec_L v$ if

- $u \in \mathsf{e\text{-}var}(q)$ and $v \in \mathsf{const}(q) \cup \mathsf{f\text{-}var}(q)$; or
- $u \in \mathsf{const}(q)$ and $v \in \mathsf{f\text{-}var}(q)$; or
- u, v are from the same set of either $\mathsf{e\text{-}var}(q)$, $\mathsf{const}(q)$ or $\mathsf{f\text{-}var}(q)$, and u is the lexicographically smaller than v.

The order $\prec_L$ is extended to sides of $\mathcal{G}(q)$ as follows: $(u \overset{R}{\rightharpoonup} v) \prec_L (u' \overset{R'}{\rightharpoonup} v')$ if *(i)* $v \prec_L v'$, or *(ii)* $v = v'$ and

- $R = P_1$ and $R' = P_2^-$ for binary predicates P_1, P_2 from q, or
- $R = P_1$ (resp. $R = P_1^-$), $R = P_2$ (resp. $R = P_2^-$) for binary predicates P_1, P_2 in q, and P_1 is lexicographically smaller than P_2,

or *(iii)* $v = v'$, $R = R'$ and $u \prec_L u'$. Figure 10.1 depicts the the internalisation sequence of each query w.r.t. $\prec_L$. As we have discussed in Example 4.11, the query shown in Figure 10.1e is not internalisable. We can also observe that the e-cycle in Figure 10.1b in the canonical graph remains in the rolled-up graph. ♢

Now we are ready to characterise answers to an internalisable query by its rolled-up graphs. Given an OWL 2 ontology and a dataset, checking answers of an internalisable query can be reduced entailment of data assertions and ontology satisfiability.

Theorem 10.19. *Let $\mathcal{O}$ be an OWL 2 ontology, let $\mathcal{D}$ be a dataset, and let $q(\mathbf{x})$ be an internalisable query, and let $G = \langle V_f, V_e, V_c, E, \rho \rangle$ be the rolled-up graph of q. Moreover, let σ be a possible answer to q in $\mathcal{O} \cup \mathcal{D}$. Then σ is a certain answer to q iff the following conditions hold.*

1. $\mathcal{O} \cup \mathcal{D} \models C(u)\sigma$ *for each* $u \in V_f \cup V_c$ *and* $C \in \rho(u)$;

2. $\mathcal{O} \cup \mathcal{D} \cup \{\bigsqcap_{C \in \rho(y)} C\sigma \sqsubseteq C_\bot\}$ *is unsatisfiable for each* $y \in V_e$; *and*

3. $\mathcal{O} \cup \mathcal{D} \models P(u, v)\sigma$ *for each edge* $u \xrightarrow{P} v \in E$.

Proof. Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be an arbitrary model of $\pi(\mathcal{O}) \cup \mathcal{D}$. To show the "only if" direction, we assume that $\sigma \in \mathsf{cert}(q, \pi(\mathcal{O}) \cup \mathcal{D})$. Then $\mathcal{I} \models q\sigma$ and by Proposition 10.12 we have $\mathcal{I} \models \mathcal{G}(q)\sigma$. By Lemma 10.14, we have $\mathcal{I} \models G\sigma$. For each $u \in V_f \cup V_c$ and $C \in \rho(u)$, $\mathcal{I} \models C(u)\sigma$ since $C(u)\sigma$ is ground. For each $u \xrightarrow{P} v \in E$, we have $P(u, v)\sigma$ is ground by Theorem 10.17. Thus, we have $\mathcal{I} \models P(u, v)\sigma$ for each edge $u \xrightarrow{P} v \in E$. Let σ' be a witness of G and σ in $\mathcal{I}$. For each $y \in V_e$ and $C \in \rho(y)$, we have $\sigma'(y) \in \Delta^\mathcal{I}$ and $\mathcal{I} \models C(\sigma'(y))\sigma$. So $\mathcal{I} \not\models \bigsqcap_{C \in \rho(y)} C\sigma \sqsubseteq C_\bot$.

We next prove the "if" direction. For each $y \in V_e$, we have $\mathcal{I}$ is not a model of $\mathcal{O} \cup \mathcal{D} \cup \{\bigsqcap_{C \in \rho(y)} C\sigma \sqsubseteq C_\bot\}$. So there exists a constant $c_y \in \Delta^\mathcal{I}$ such that $c_y \in (\bigsqcap_{C \in \rho(y)} C\sigma)^\mathcal{I}$. So $c_y \in C^\mathcal{I}$ for each $C \in \rho(y)$. Let σ' be the substitution that maps each $y \in V_e$ to c_y. Then σ' is a witness of G and σ in $\mathcal{I}$ and hence $\mathcal{I} \models G\sigma$. By Lemma 10.14, we have $\mathcal{I} \models q\sigma$. Therefore, $\sigma \in \mathsf{cert}(q, \pi(\mathcal{O}) \cup \mathcal{D})$. □

Let $\mathcal{L}$ be a DL language, and let q be an internalisable query. We say that q is internalisable in $\mathcal{L}$ if there exists an ordering $\prec$ such that the rolled-up graph $G = \langle V_f, V_e, V_c, E, \rho \rangle$ of q w.r.t. $\prec$ satisfies the following conditions.

- For each $u \in V_f$, $\rho(u)$ is set of $\mathcal{L}$-concepts.
- For each $y \in V_e$, $\bigsqcap_{C \in \rho(y)} C \sqsubseteq C_\bot$ is a GCI in $\mathcal{L}$.

Example 10.20. Let $\sigma = \{x \mapsto a_x, z \mapsto a_z\}$ be a possible answer to the query shown in Figure 10.1a w.r.t. a ontology $\mathcal{O}$ and a dataset $\mathcal{D}$. Then σ is an answer iff $\mathcal{O} \cup \mathcal{D} \models \{P(a_x, a_z), \exists R_1 \exists R_3.\{a_z\}(a_x), \exists R_4 \exists R_2.\{a_x\}(a_z)\}$ and $\mathcal{O} \cup \mathcal{D} \cup \{A \sqsubseteq C_\bot\}$ is unsatisfiable. ♢

Therefore, the decidability of answering internalisable queries over OWL 2 ontologies is implied by the decidability of entailment checking of data assertions and satisfiability checking.

Part III

The PAGOdA System

Chapter 11

System Description

We have implemented our approach in a system called PAGOdA, which is written in Java and is available under an academic license. Our system integrates the materialisation-based datalog reasoner RDFox (Motik et al., 2014) and the fully-fledged OWL 2 reasoner HermiT (Glimm et al., 2014), and we also exploit the combined approach for $\mathcal{ELHO}^r_\bot$ (see Section 7.2) implemented in KARMA.[1] PAGOdA exploits all improvements and optimisations described in Part II; the realisation of our approaches is discussed in this chapter.

PAGOdA accepts as input arbitrary OWL 2 ontologies, datasets in turtle format[2] and queries in an SPARQL-style syntax, which we specify next.

11.1 Query Language of PAGOdA

PAGOdA accepts as input queries in a SPARQL-style syntax. We deviate from the standard SPARQL syntax and semantics for the following reasons.

- PAGOdA was designed to support arbitrary conjunctive queries. As already discussed in Section 4.2, conjunctive queries that do not correspond to OWL 2 ontologies are not legal according to the OWL 2 entailment regime. Although PAGOdA is not guaranteed to be complete for all of such queries due to a limitation of the underlying OWL 2 reasoner, it is capable to fully answer some of such queries if the computed lower and upper bounds coincide. Furthermore, PAGOdA is also guaranteed to be complete for arbitrary conjunctive queries if an input ontology is in OWL 2 RL or $\mathcal{ELHO}^r_\bot$.

[1] KARMA is available at `http://www.cs.ox.ac.uk/isg/tools/KARMA/`.
[2] `http://www.w3.org/TR/turtle/`

- We also aim at providing a flexible mechanism at the syntax level to specify which variables in a query should be interpreted under the first-order semantics or the ground semantics.

Syntax A SPARQL conjunctive query Q is *admissible* if it is of the form

$$\texttt{SELECT } ?x_1 \ldots ?x_n \texttt{ WHERE } \{ \; s_1 \; p_1 \; o_1 \; . \quad \ldots \; s_m \; p_m \; o_m \; \} \tag{11.1}$$

where *(i)* each p_i is an IRI, *(ii)* o_i is an IRI whenever p_i is rdf:type, *(iii)* no *name* exists such that both $?name$ and $\_:name$ occur in the query, and *(iv)* no IRI in the query with the only exception of rdf:type is in the reserved vocabulary of RDF, RDFS or OWL.

Conditions *(i)* and *(ii)* ensure that variables do not occur in the places of predicates, so that each triple pattern in the BGP of an admissible query directly corresponds to a first-order atom. Condition *(iii)* guarantees that a variable and a blank node in Q correspond to different first-order variables.

Example 11.1. Consider the legal queries in Example 4.13. Only query Q1 is admissible for PAGOdA. Queries Q2 and Q3 are not admissible since they contain IRIs from the reserved vocabulary of RDFS and OWL other than rdf:type; query Q4 is also not admissible as it contains a variable in a place of predicates. ♢

Semantics We define a translation function Γ from RDF terms and query variables to first-order terms as follows.

$$\Gamma(t) = \begin{cases} x & \text{if } t \text{ is a variable of the form } ?x, \\ y & \text{if } t \text{ is a blank nodes of the form } \_:y, \\ t & \text{otherwise.} \end{cases}$$

The translation function extends to triple patterns in admissible queries as follows.

$$\Gamma(s_i \; p_i \; o_i) = \begin{cases} o_i(\Gamma(s_i)) & \text{if } p_i \text{ is rdf:type}, \\ p_i(\Gamma(s_i), \Gamma(o_i)) & \text{otherwise.} \end{cases}$$

The application of the translation function extends to sets naturally. Let Q be an admissible query for PAGOdA of the form (11.1), and let V, B be the sets of query variables and blank nodes in Q, respectively. The vector of *answer variables* $\mathbf{x}_Q$ of Q is defined as $(x_1, \ldots, x_n)$; the vector of *ground variables* $\mathbf{z}_Q$ of Q is defined as a vector $(u_1, \ldots, u_m)$ such that $\{u_1, \ldots, u_m\} = \Gamma(V) \setminus \mathbf{x}_Q$; and the vector of *existential*

variables $\mathbf{y}_Q$ is defined as a vector $(y_1, \ldots, y_\ell)$ such that $\{y_1, \ldots, y_\ell\} = \Gamma(B)$. Then the query q_Q corresponding to Q is defined as the following conjunctive query.

$$q_Q(\mathbf{x}_Q, \mathbf{z}_Q) = \exists \mathbf{y}_Q [\bigwedge_{i=1}^{n} \Gamma(s_i \ p_i \ o_i)].$$

The set of answers to Q in a knowledge base $\mathcal{K}$ computed by PAGOdA is the set of projections on $\mathbf{x}_Q$ of certain answers of q_Q w.r.t. $\mathcal{K}$.

The semantics of an admissible query in PAGOdA coincides with the standard SPARQL semantics if no blank nodes appear in the query, but both semantics differ in the presence of blank nodes. PAGOdA treats blank nodes as existentially quantified variables in conjunctive queries, which can be mapped to arbitrary domain elements in models of an input knowledge base.

Example 11.2. Consider again the legal queries in Example 4.13, the semantics of Q1 in PAGOdA is exactly the same as the standard SPARQL semantics. In contrast, the following query Q4 differs both semantics.

$$\texttt{SELECT } ?x \texttt{ WHERE } \{ \ ?x \ \mathsf{eats} \ \_{:}y \ . \quad \_{:}y \ \mathsf{rdf{:}type} \ \mathsf{Plant} \ \} \tag{Q4}$$

The semantics of Q4 in PAGOdA is based on the first-order semantics of $q_{Q4}(x) = \exists y[\mathsf{eats}(x, y) \wedge \mathsf{Plant}(y)]$, and hence it captures the standard SPARQL semantics of Q2, which is not admissible for PAGOdA. ♢

This syntax and semantics are flexible enough to express arbitrary conjunctive queries. Moreover, in this way, we can capture the ground semantics of conjunctive queries, the first-order semantics or a mixture of both. As mentioned before in Chapter 6, the class of queries that can be fully answered by PAGOdA depends on the capability of the underlying OWL 2 reasoner. Let Q be an admissible query, and fix an arbitrary input knowledge base. Using HermiT as the underlying OWL 2 reasoner, PAGOdA's behaviour for Q is listed as follows.

- If q_Q is internalisable, PAGOdA is sound and complete for Q.
- If q_Q is not internalisable but the computed lower and upper bounds for Q coincide, PAGOdA is sound and complete for Q.
- If q_Q is not internalisable and the computed bounds for Q differ, PAGOdA is able to provide a set of sound answers and an upper bound for query answers to estimate how incomplete the sound answers are.

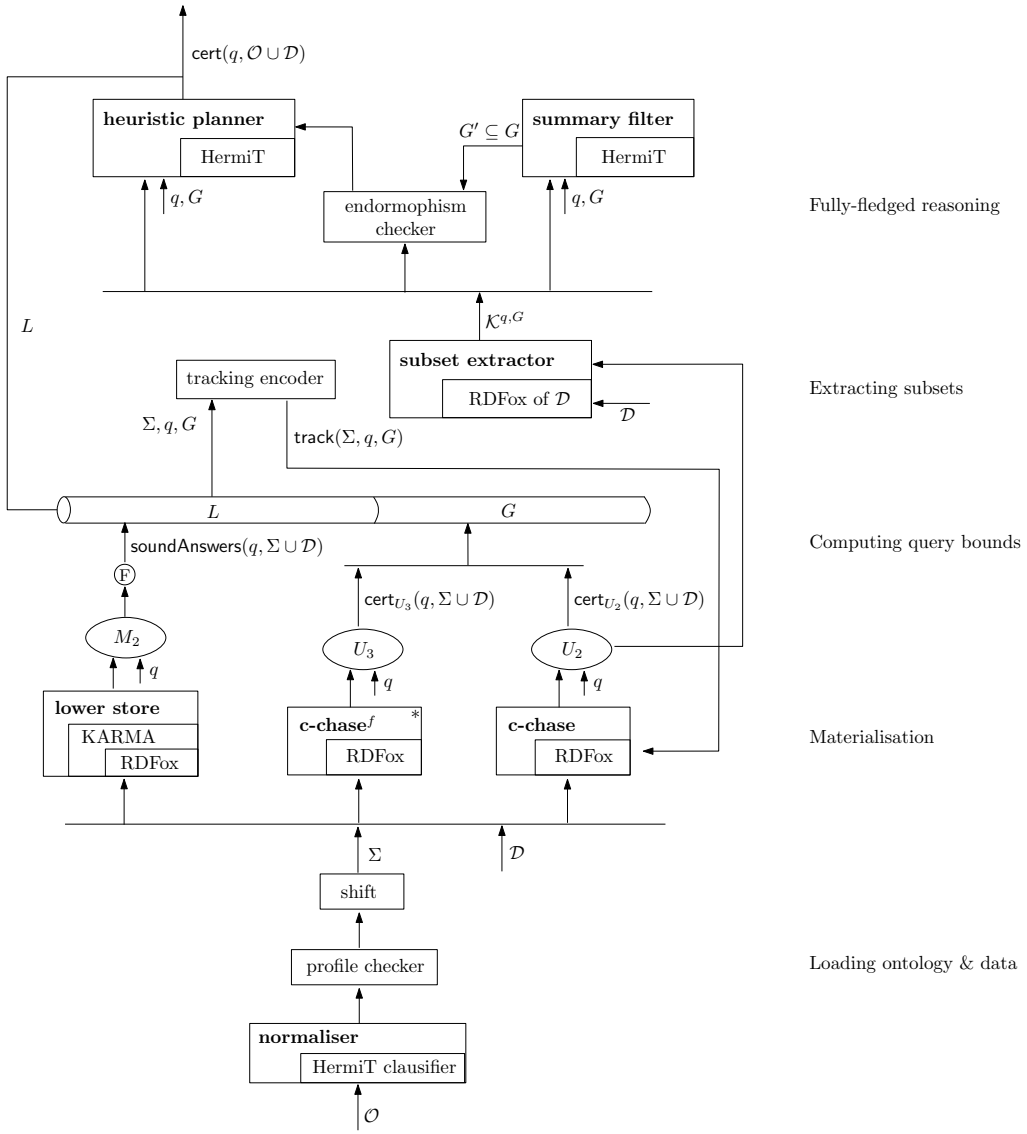


Figure 11.1: The architecture of PAGOdA

11.2 Architecture of PAGOdA

The architecture of PAGOdA is depicted in Figure 11.1. Each box in Figure 11.1 represents a component of PAGOdA, and indicates any external systems that are exploited within that component. We could, in principle, use any materialisation-based datalog reasoner that supports conjunctive query evaluation and the incremental addition of facts, and any fully-fledged OWL 2 reasoner that supports entailment checking of data assertions and satisfiability checking.

PAGOdA uses four instances of RDFox (one in each of the lower bound, c-chase, c-chase^f and subset extractor components) and two instances of HermiT (one in each

of the summary filter and dependency graph components).

The process of fully answering a query can be divided into several steps. Here, we distinguish between query independent steps and query dependent ones. As we can see in Figure 11.1, the "loading ontology and data" and "materialisation" steps are query independent. Therefore, both of them are counted as *pre-processing* steps. "Computing query bounds", "extracting subset" and "fully-fledged reasoning" are query dependent, and are called *query processing* steps. We next describe the each component, following the process flow of PAGOdA.

Loading Ontology and Data PAGOdA uses the OWL API to parse the input ontology $\mathcal{O}$. The dataset $\mathcal{D}$ is given separately in turtle format. The normaliser then computes the set of rules corresponding to the axioms in the ontology. PAGOdA's normaliser is an extension of HermiT's clausification component (Glimm et al., 2014), which transforms axioms into so-called DL-clauses (Motik et al., 2009c). The dataset is loaded directly into (the four instances of) RDFox.

After normalisation, the ontology is checked to determine if it is inside OWL 2 RL or $\mathcal{ELHO}^r_\bot$. If an input ontology is in OWL 2 RL (resp. $\mathcal{ELHO}^r_\bot$), then RDFox (resp. KARMA) is already sound and complete, and in such cases PAGOdA simply processes the ontology, dataset and queries using the relevant component. Otherwise, PAGOdA uses a dedicated program shifting component to enrich the deterministic part of the ontology with additional information from disjunctive rules (see Section 7.1), resulting in a set of rules Σ.

Materialisation There are three components involved in this step, namely lower bound, c-chase and c-chase^f. Each of these takes as input Σ and $\mathcal{D}$, and each computes a materialisation (shown in Figure 11.1 as ellipses). The lower bound component performs Step 1 and 2 from Section 7.3 in order to compute an aggregated lower bound M_2^L. The c-chase and c-chase^f components compute the M_2^U and M_3^U upper bound materialisations as described in Section 8.2 and 8.3 using a dedicated implementation of the c-chase algorithm. The chase sequence is stored in RDFox, and the applicability of existential and disjunctive rules is determined by posing SPARQL queries to RDFox. When applying a disjunctive rule (while computing M_3^U), PAGOdA uses a choice function to select one of the disjuncts. As discussed in Section 8.4, the choice function should try to select disjuncts that will not (eventually) lead to a contradiction. To this end, PAGOdA implements the following heuristics.

- We construct a standard dependency graph containing an edge from predicate P to Q if there is a rule where P occurs in the body and Q in the head. Then, we compute a preference ordering on the predicates occurring in a disjunction according to their distance from $\bot$ in the dependency graph, preferring those that are furthest from $\bot$.

- We exploit the result of materialising $\mathcal{D}$ using the shifting enriched rules in Σ (see Section 7.1). If a fact of the form $\overline{P}(\mathbf{a})$ is obtained in the materialisation, then $\neg P(\mathbf{a})$ follows from the knowledge base. Hence, if we have obtained $\overline{P}(\mathbf{a})$, then we try to avoid choosing $P(\mathbf{a})$ from a disjunct $P(\mathbf{a}) \vee \chi$ during chase computation.

If M_2^L contains a contradiction, then the input ontology and dataset is unsatisfiable, and PAGOdA reports this and terminates. If $\bot_s$ is derived in M_3^U, then the computation of the materialisation is aborted and M_3^U is no longer used. If M_2^U contains $\bot_s$, then PAGOdA checks the satisfiability of $\Sigma \cup \mathcal{D}$; in effect, it computes $\mathsf{cert}(\bot, \Sigma \cup \mathcal{D})$. If the answer to this query is non-empty, then the input ontology and dataset is unsatisfiable, and PAGOdA reports this and terminates; otherwise the input ontology and dataset is satisfiable, and PAGOdA is able to answer queries.

Computing Query Bounds Let Q be an admissible query for PAGOdA. PAGOdA computes the lower bound answers L^Q of Q obtained by first evaluating $q_Q(\mathbf{x}_Q, \mathbf{z}_Q)$ in the lower bound materialisation M_2^L and then projecting its answers to $\mathbf{x}_Q$. In order to do this it exploits KARMA's implementation of the filtration procedure (algorithm soundAnswers in Section 7.2), but for clarity this step is shown separately (as a circle with an 'F' in it) in Figure 11.1. If $\bot_s$ was not derived when computing the M_3^U materialisation, the upper bound answers to Q is obtained by $U^Q = \mathsf{cert}(q_Q, M_2^U) \cap \mathsf{cert}(q_Q, M_3^U)$; otherwise $U^Q = \mathsf{cert}(q_Q, M_2^U)$. In either case U^Q is computed directly by using RDFox to answer the conjunctive query q_Q w.r.t. the relevant materialisation. The gap between lower and upper bound query answers to the input query is defined as $G^Q = \{\sigma \in U^Q \mid \sigma|_{\mathbf{x}_Q} \notin L^Q\}$.

Extracting Subsets The tracking encoder component implements the datalog encoding based on Definition 9.1 with the optimisations described in Section 9.3. The resulting datalog knowledge base of $\mathcal{K}$, q_Q and G^Q is added to the rules and facts in the c-chase component, and RDFox is used to extend the c-chase materialisation accordingly. The freshly derived facts (over the tracking predicates introduced by

the tracking encoder) are then passed to the subset extractor component, which uses these facts to identify all the facts and rules that are relevant for checking gap answers, and computes the intersection between relevant facts and the input dataset $\mathcal{D}$ by querying an instance of RDFox containing $\mathcal{D}$ only.

Fully-fledged Reasoning PAGOdA uses HermiT to verify gap answers in $G^{\mathcal{Q}}$. As HermiT is capable of entailment checking of data assertions and satisfiability checking over OWL 2 ontologies, we have also implemented the rolling-up technique in Section 10.2 to transform answer checking of internalisable queries into entailment checking of data assertions and satisfiability checking. In the summary filter component, PAGOdA uses HermiT to filter out gap answers that are not entailed by a summary of $\mathcal{K}^q$ (see Section 10.1.1). The remaining gap answers $G' \subseteq G^{\mathcal{Q}}$ are then passed to the endormorphism checker, which exploits a greedy algorithm to compute an (incomplete) dependency graph between answers in G'. This graph is used by the heuristic planner to optimise the order in which the answers in G' are checked using HermiT (see Section 10.1.1). The projections on $\mathbf{x}_Q$ of verified answers from G' and the lower bound $L^{\mathcal{Q}}$ are returned as the answers to Q.

Chapter 12

Evaluation

We have evaluated our query answering system PAGOdA on a wide range of realistic and benchmark ontologies, datasets and queries, and we have compared its performance with state-of-the-art query answering systems. Our test data and the systems used for comparison are introduced in Section 12.1 and 12.2, respectively. Our results are discussed in Section 12.3. Experiments were conducted on a 32 core 2.60GHz Intel Xeon E5-2670 with 250GB of RAM, and running Fedora 20. All test ontologies, queries, and results are available online.[1]

The empirical results have shown that PAGOdA outperforms the state-of-the-art query answering systems in different aspects including the quality of the computed answers and the efficiency to compute answers. Moreover, the scalability test on systems has revealed that the processing time for PAGOdA to answer queries appears to be polynomial in most of the cases. Finally, we have shown that each individual technique proposed in the thesis does make impact in the experiment.

12.1 Test Ontologies and Queries

Table 12.1 summarises our test data. The first two columns in the table indicate the total number of DL axioms in each test ontology as well as the total number of rules after normalisation. We are interested in ontologies that are not captured by OWL 2 RL and hence cannot be fully processed by RDFox; thus, the number of rules containing existential quantifications and disjunctions is especially relevant and is given in the third and fourth columnns of the table, respectively. Finally, the rightmost column lists the number of facts in each dataset.

[1] `http://www.cs.ox.ac.uk/isg/tools/PAGOdA/2015/jair/`

	♯axioms	♯rules	♯∃-rules	♯∨-rules	♯facts
LUBM(n)	93	133	15	0	$n \times 10^5$
UOBM(n)	186	234	23	6	$2.6n \times 10^5$
FLY	14,447	18,013	8396	0	8×10^3
NPD	771	778	128	14	3.8×10^6
DBPedia$^+$	1,716	1,744	11	5	2.9×10^7
ChEMBL	2,593	2,960	426	73	2.9×10^8
Reactome	559	575	13	23	1.2×10^7
Uniprot	442	459	20	43	1.2×10^8

Table 12.1: Statistics for test datasets

LUBM and UOBM are widely-used reasoning benchmarks (Guo et al., 2005; Ma et al., 2006). The ontology axioms in these benchmarks have been manually created and are considered fixed, whereas the data is synthetically generated according to a parameter n that determines its size. LUBM and UOBM come with 14 and 15 standard queries, respectively. To make the tests on LUBM more challenging, we extended the benchmark with 10 additional queries for which datalog lower-bound answers are not guaranteed to be complete (as is the case for the standard queries).

FLY is a realistic ontology that describes the anatomy of the Drosophila and which is currently integrated in the Virtual Fly Brain tool.[2] Although the data is rather small compared to other test cases (about $8,000$ facts), the ontology is very rich in existentially quantified rules, which makes query answering especially challenging. We tested 6 realistic queries that were provided by the developers of the ontology.

NPD FactPages is an ontology describing the petroleum activities in the Norwegian continental shelf. The ontology comes with a realistic dataset containing 3.8 million facts. Unfortunately, for NPD we have no realistic queries so we tested all atomic queries over the signature of the ontology.

DBPedia contains information about Wikipedia entries. Although the dataset is rather large, the ontology axioms are simple and can be captured by OWL 2 RL. To provide a more challenging test, we have used the ontology matching system LogMap (Jiménez-Ruiz and Cuenca Grau, 2011) to extend DBPedia with a tourism ontology containing both existential and disjunctive rules. As in the case of NPD we have no example test queries, so we focused our evaluation on atomic queries.

[2] `http://www.virtualflybrain.org/site/vfb_site/overview.htm`

ChEMBL, Reactome, and Uniprot are realistic ontologies that have been made publicly available through the European Bioinformatics Institute (EBI) linked data platform.[3] These ontologies are especially interesting for testing purposes. On the one hand, both the ontology axioms and data are realistic and are being used in a number of applications; on the other hand, the ontologies are rich in both existentially quantified and disjunctive rules, and the datasets are extremely large. Furthermore, the EBI website provides a number of example queries for each of these ontologies. In order to test scalability on these datasets as well as to compare PAGOdA with other systems we implemented a data sampling algorithm based on random walks (Leskovec and Faloutsos, 2006) and computed subsets of the data of increasing size. We have used for evaluation those example queries that correspond to conjunctive queries as well as all atomic queries over the relevant signature.

12.2 Comparison Systems

We compared PAGOdA against four ontology reasoners: HermiT (v.1.3.8), Pellet (v.2.3.1), TrOWL-BGP (v.1.2), and Hydrowl (v.0.2). With the single exception of TrOWL, all these systems implement sound and complete algorithms for standard reasoning tasks over OWL 2 ontologies, including ontology satisfiability checking and concept instance retrieval. Additionally, all but HermiT provide support to answer SPARQL queries.

As pointed out in the Chapter 5, there are many other systems that can answer queries over ontologies. However, these systems have generally been designed for specific fragments of OWL 2, and are incomplete for ontologies outside these fragments. Although TrOWL is also incomplete for OWL 2, it has been included in the evaluation because it is, on the one hand, a widely-used system in semantic web applications and, on the other hand, it is similar to PAGOdA in that it exploits ontology approximation techniques. In what follows, we describe the capabilities of these systems in more detail.

HermiT is a fully-fledged OWL 2 reasoner based on the hypertableau calculus (Motik et al., 2009c; Glimm et al., 2014). HermiT focuses on standard reasoning tasks in DLs. It does not provide a query answering API, but it is capable of answering (complex) concept queries and checking entailment of data assertions.

[3] `http://www.ebi.ac.uk/rdf/platform`

Pellet is a tableau-based OWL reasoner with support for conjunctive query answering (Sirin et al., 2007). Pellet provides a SPARQL API, and hence it can compute the set of all ground answers to arbitrary conjunctive queries. Moreover, Pellet is also capable of computing certain answers to tree-shaped conjunctive queries by a rolling-up technique.

TrOWL is a system based on approximated reasoning. It accepts as input an arbitrary OWL 2 ontology and a conjunctive query in SPARQL, and aims at computing ground answers to the given query (Thomas et al., 2010). TrOWL exploits a technique that approximates the input ontology into the OWL 2 QL profile, and it does not provide completeness guarantees.

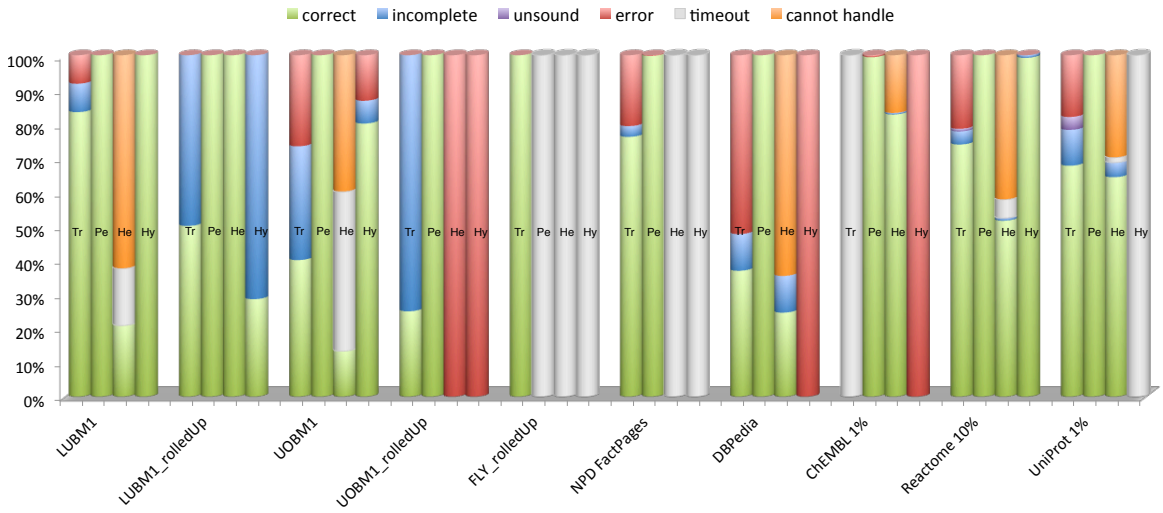
Hydrowl (Stoilos, 2014b) is a hybrid reasoning system that is similar in spirit to PAGOdA (see Section 5.2.5 for a detailed comparison). Hydrowl integrates the triple store OWLim and the OWL 2 reasoner HermiT. It accepts as input an arbitrary OWL 2 ontology and conjunctive queries as datalog rules, and then computes the ground answers to a given query.

12.3 Experiments and Results

We have performed three different sets of experiments. In the first set, we compared PAGOdA with the above mentioned systems, with respect to both the *quality* of their answers (i.e., the number of correctly answered queries) and their *performance* relative to PAGOdA. In the second set, we evaluated the *scalability* of PAGOdA by considering datasets of increasing size. Finally, in the last set, we evaluated the *effectiveness* of each of the different reasoning techniques implemented in PAGOdA.

12.3.1 Comparison with Other Systems

We have compared PAGOdA with the other systems on our test ontologies. We used LUBM(1) and UOBM(1) since they are already rather hard for some of the systems. Similarly, we used relatively small samples of the EBI platform ontologies (1% of the data for ChEMBL and UniProt, and 10% for Reactome) that can be processed by the majority of systems. For each test ontology we computed all ground answers to the corresponding test queries, and whenever possible we used the rolling-up technique (see Definition 10.13) to additionally compute all certain answers of tree-shaped queries. In the case of FLY, all test queries yield an empty set of ground



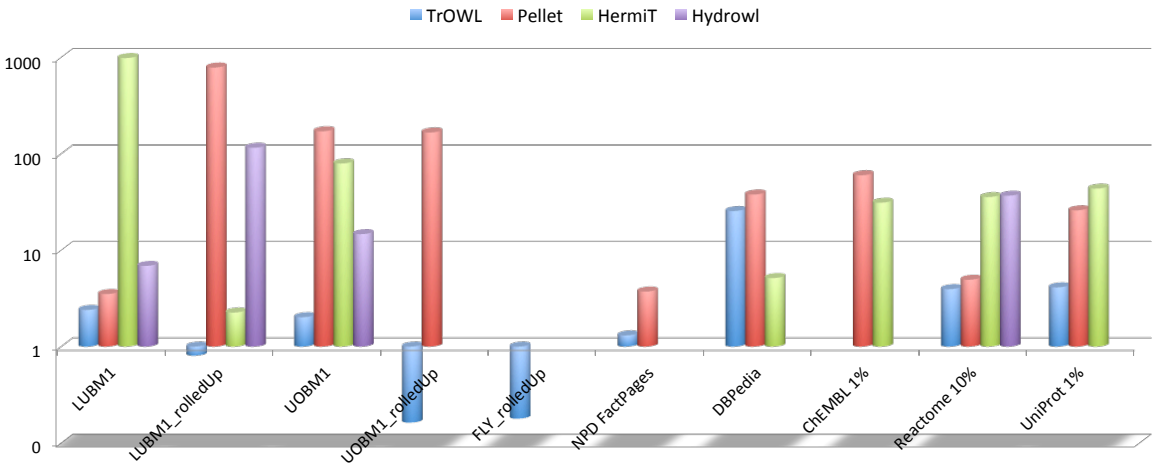
The four bars for each ontology represent Trowl, Pellet, HermiT and Hydrowl respectively.

Figure 12.1: Quality of the answers computed by each system

answers, so in this case we computed only the certain answers (all FLY queries are tree-shaped). We set timeouts of 20 minutes for answering each individual query, and 5 hours for answering all the queries over a given ontology.

Figure 12.1 summarises the quality of the answers computed by each reasoner. Each bar in the figure represents the performance of a particular reasoner w.r.t. a given ontology and set of test queries. We use green to indicate the percentage of queries for which the reasoner computed all the correct answers, where correctness was determined by majority voting, and blue (resp. purple) to indicate the percentage of queries for which the reasoner was incomplete (resp. unsound). Red, orange and grey indicate, respectively, the percentage of queries for which the reasoner reported an exception during execution, did not accept the input query, or exceeded the time-out. Under our criterion of correctness, PAGOdA was able to correctly compute all answers for every query and test ontology within the given timeouts. Consequently, the performance of PAGOdA is not represented in the figure.

Figure 12.2 summarises the performance of each system relative to PAGOdA, but in this case we considered only those queries for which the relevant system yields an answer (even if the computed answer is unsound and/or incomplete). This is not ideal, but we chose to consider all such queries (rather than only the queries for which the relevant system yields the correct answer) because *(i)* the resulting time measurement is obviously closer to the time that would be required to correctly answer all queries; and *(ii)* correctness is only relative as we don't have a gold standard for query answers. For each ontology and reasoner, the corresponding bar shows t_2/t_1 (on



Each bar depicts the total time to answer all queries for an ontology in comparison with PAGOdA.

Figure 12.2: Performance comparison with other systems

a logarithmic scale), where t_1 (resp. t_2) is the total time required by PAGOdA (resp. the compared system) to compute the answers to the queries under consideration; a missing bar indicates that the comparison system failed to answer any queries within the given timeout. Please note that two different bars for the same ontology are not comparable as they may refer to different sets of queries, so each bar needs to be considered in isolation.

We can draw the following conclusions from the results of our experiments.

- TrOWL is faster than PAGOdA on LUBM with rolling up, UOBM with rolling up and FLY with rolling up, but it is incomplete for 7 out of 14 LUBM queries and 3 out of 4 UOBM queries. For ChEMBL, TrOWL exceeds the timeout while performing the satisfiability check. For the remaining ontologies, PAGOdA is more efficient in spite of the fact that TrOWL is incomplete for some queries, and even unsound for several UniProt queries.

- Pellet is one of the most robust systems in our evaluation. Although it times out for the FLY ontology, it succeeds in computing all answers in the remaining cases. We can observe, however, that in all cases Pellet is significantly slower than PAGOdA, sometimes by more than two orders of magnitude.

- HermiT can only answer queries with one free variable, so we could not evaluate atomic binary queries. We can see that HermiT exceeds the timeout in many cases. In the tests where HermiT succeeds, it is significantly slower than PAGOdA.

- Although Hydrowl is based on a theoretically sound and complete algorithm, it was found to be incomplete in some of our tests. It also exceeded the timeout on all queries for three of the ontologies, ran out of memory on all queries for another two of the ontologies, and reported an exception for ChEMBL 1%. In the remaining cases, it was significantly slower than PAGOdA.

12.3.2 Scalability Tests

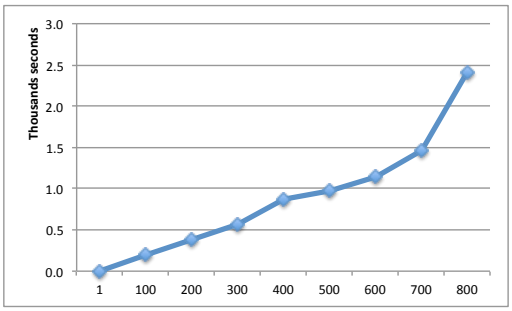
We tested the scalability of PAGOdA on LUBM, UOBM and the ontologies from the EBI linked data platform. For LUBM we used datasets of increasing size with a step of $n = 100$. For UOBM we also used increasingly large datasets with step $n = 100$ and we also considered a smaller step of $n = 5$ for hard queries. Finally, in the case of EBI's datasets, we implemented a data sampling algorithm based on random walks and computed subsets of the data of increasing sizes from 1% of the original dataset up to 100% in steps of 10%. We used the test queries described in Section 12.1 for each of these ontologies; as in Section 12.3.1, we computed ground answers and, whenever possible, used the rolling-up technique to additionally compute certain answers. For each test ontology we measured the following:

- *Pre-processing time.* This includes all pre-processing steps in Chapter 11 as well as satisfiability checking (i.e., query processing for the Boolean unsatisfiability query).

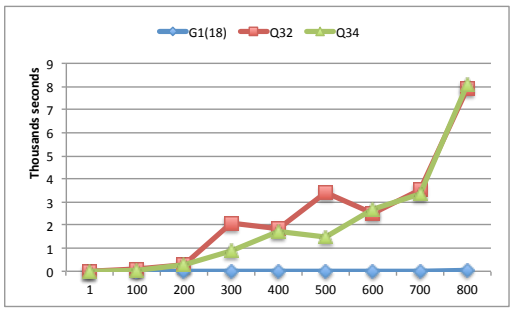
- *Query processing time.* This is the time to perform the query processing steps for a query in the given ontology. We organise the test queries into the following three groups depending on the techniques exploited by PAGOdA to compute their answers:

  - **G1**: queries for which the lower and upper bounds coincide;
  - **G2**: queries with a non-empty gap, but for which summarisation is able to filter out all remaining candidate answers; and
  - **G3**: queries where the fully-fledged reasoner is called over an ontology subset on at least one of the test datasets.

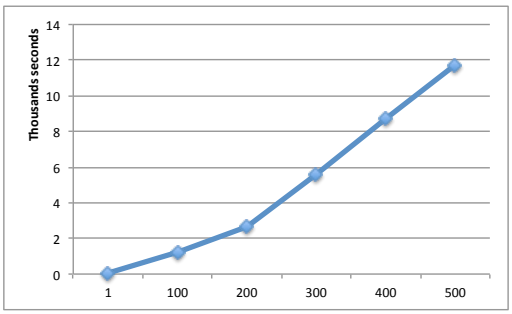
In the scalability test, we set a timeout of 5 hours for answering all queries and 2.5 hours for each individual query. For LUBM and UOBM, we increased the size of the dataset until PAGOdA exceeded the timeout; for the other ontologies, PAGOdA was able to answer all queries within the timeout, even with the largest dataset.



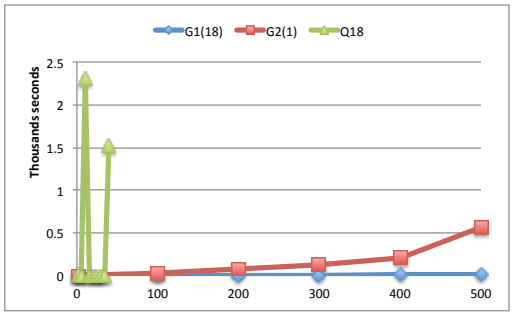
(a) LUBM pre-processing



(b) LUBM query processing



(c) UOBM pre-processing



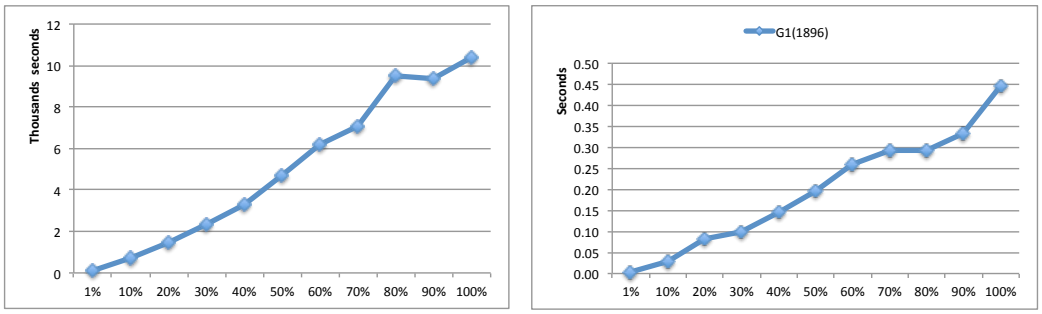
(d) UOBM query processing

Figure 12.3: Scalability tests on benchmarks

Pellet was the only compared system found to be sound and complete for our test ontologies and queries, so we have also conducted scalability tests on it. The scalability of Pellet is, however, limited: it already failed on LUBM(100), UOBM(5), as well as ChEMBL 10% and Uniprot 10%. The only dataset were Pellet managed to process at least two data samples was Reactome, where it succeeded on all samples smaller than 40%. The case for Reactome is discussed in detail later on.

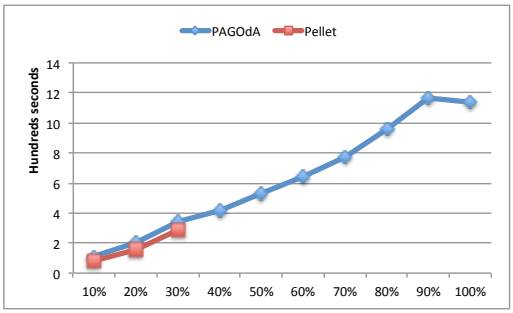
Our results are summarised in Figure 12.3 and 12.4. For each ontology, we plot time against the size of the input dataset, and for query processing we distinguish different groups of queries as discussed above. PAGOdA behaves relatively uniformly for queries in **G1** and **G2**, so we plot only the average time per query for these groups. In contrast, PAGOdA's behaviour for queries in **G3** is quite variable, so we plot the time for each individual query.

LUBM(n) As shown in Figure 12.3a, pre-processing is fast, and times appear to scale linearly with increasing dataset size. All LUBM queries belong to either **G1** or **G3** with the latter group containing just two queries. Figure 12.3b illustrates the average query processing time for queries in **G1**, which never exceeds 13 seconds, as

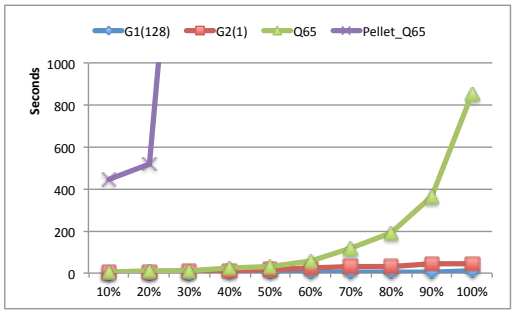


(a) ChEMBL pre-processing

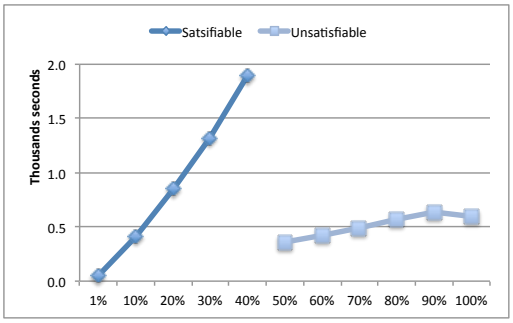
(b) ChEMBL query processing



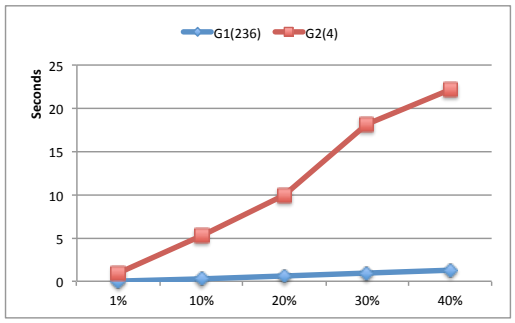
(c) Reactome pre-processing



(d) Reactome query processing



(e) Uniprot pre-processing



(f) Uniprot query processing

Figure 12.4: Scalability tests on EBI linked data platform

well as the time for each of the two queries in **G3** (Q32 and Q34), which reaches 8,000 seconds for LUBM(800), most of which is accounted for by HermiT.

UOBM(n) processing times never exceeds 8 seconds from UOBM(1) to UOBM(500). We found one query in **G2**. Processing times for this query were somewhat longer than for those in **G1** and reached 569s for UOBM(500). Finally, we found one query (Q18) that, due to UOBM's randomised data generation, was in different groups for different datasets: in UOBM(1), UOBM(10) and UOBM(50) it was in **G3**, and HermiT was called on the relevant subsets to fully answer the query; in UOBM(40) it was in **G2**, and HermiT was called on only the summary of the relevant subset; and

in all the remaining cases shown in Figure 12.3d it was in **G1**, and the lower and upper bounds coincided. This query timed out in UOBM(50), due to the time taken by HermiT to reason over the relevant subset, but we have shown the times for the remaining **G1** and **G2** queries up to UOBM(500).

We next take a closer look at the most difficult query in **G3** of UOBM. The query is $q(x) = \mathsf{PeopleWithManyHobbies}(x)$. The concept $\mathsf{PeopleWithManyHobbies}$ appears in an equivalence axiom $\mathsf{PeopleWithManyHobbies} \equiv \geq 3\,\mathsf{like}$. The translation of this axiom into rules contains in the following disjunctive rule.

$$\mathsf{like}(x, y_1) \wedge \mathsf{like}(x, y_2) \wedge \mathsf{like}(x, y_3) \rightarrow \mathsf{PeopleWithManyHobbies}(x) \vee y_1 \approx y_2 \vee y_1 \approx y_3 \vee y_2 \approx y_3$$

In the upper bound computation with the refined handling for disjunctive rules, the choice function always prefer $\mathsf{PeopleWithManyHobbies}(x)$ over qualities atoms. The reason is that it is very difficult to choose the correct equality atoms to avoid deriving $\bot_s$, so it is a conservative choice to choose $\mathsf{PeopleWithManyHobbies}(x)$. However, this results in a large number of spurious answers for this query in the upper bound. A potential solution is to look into the choice function and design a more advanced one, which is left as a future work.

ChEMBL As shown in Figure 12.4a, pre-processing times are significant but manageable, and again appear to scale linearly with dataset size. All test queries were contained in **G1**. Figure 12.4b illustrates the average processing times for all queries, which was less than 0.5s for all datasets and increases smoothly with dataset size.

Reactome As shown in Figure 12.4c, pre-processing times again appear to scale quite smoothly. Groups **G2** and **G3** each contained one query, with all the remaining queries belonging to **G1**. Query processing times are shown in Figure 12.4d. Average query processing time for queries in **G1** never exceeded 10 seconds. Average processing times for **G2** queries appeared to grow linearly to the size of datasets, and average time never exceeded 10 seconds. Finally, it can be seen that the **G3** query (Q65) is much more challenging, but it could still be answered in less than 900 seconds, even for the largest dataset.

As already mentioned, we also tested the scalability of Pellet on Reactome, where Pellet is able to process the samples of size $10\%, 20\%$ and 30%. The pre-processing time of Pellet on these datasets is comparable with PAGOdA as shown in Figure 12.4c. Average query-processing times for queries in **G1** and **G2** are slightly higher than

	LUBM(100)	UOBM(1)	FLY	NPD	DBPedia
Total	35	20	6	478	1247
$L_1 + U_1$	26	4	0	442	1240
$L_2 + U_1$	33	4	5	442	1241
$L_2 + U_2$	33	12	5	442	1241
$L_2 + U_{2\|3}$	33	16	5	473	1246

ChEMBL 1%	Reactome 10%	Uniprot 1%
1896	130	240
1883	82	204
1883	82	204
1883	98	204
1896	128	236

Table 12.2: ♯Queries answered by different bounds

those of PAGOdA. In contrast, times for query Q65 were significantly higher: 445s, 518s and 2,626s for Reactome 10%, 20% and 30%, respectively (see Figure 12.4d). Processing times for Q65 in PAGOdA, however, grow smoothly thanks to the effectiveness of the subset extraction technique, which is able to keep the input to the fully-fledged reasoner small, even for the largest datasets.

Uniprot In contrast to the other cases, Uniprot as a whole is unsatisfiable; our sampling technique can, however, produce a satisfiable subset. Figure 12.4e illustrates pre-processing times. As can be seen, these drop abruptly for unsatisfiable samples (50% and larger); this is because unsatisfiability can be efficiently detected in the lower bound. The figure shows that time to detect inconsistency for 100% is even less than that for 90%; this is because the time is dominated by loading time, and I/O performance varies from run to run. Query processing times were only considered for satisfiable samples (see Figure 12.4f). There were no queries in **G3**, and only four in **G2**. We can observe that average times for all queries appear to scale linearly with data size for both groups.

12.3.3 Effectiveness of the Implemented Techniques

We have evaluated the effectiveness of the various reasoning techniques implemented in PAGOdA by comparing the numbers of test queries that can be fully answered using the relevant technique.

Query Bounds In Chapter 7 and 8 we described different techniques for computing lower and upper bound query answers. Table 12.2 illustrates the effectiveness of each of these bounds in terms of the number of queries for which the bounds coincided on our test ontologies. In the table, we refer to the lower bound described in Section 7.1 as L_1 and to the aggregated lower bound described in Section 7.3 as L_2. Similarly, we refer to the three upper bound computation techniques discussed in Section 8.4 as U_1, U_2, U_3 and the combined upper bound $U_{2|3}$. We can observe the following from our experiments:

- The basic lower and upper bounds suffice to answer most of the queries in many test ontologies. In particular, L_1 and U_1 matched in 26 out of the 35 queries for LUBM(100), 442 out of 478 for NPD, 240 out of 1247 for DBPedia, 1883 out of 1896 for ChEMBL, and 204 out of 240 for Uniprot.

- The aggregated lower bound L_2 was very effective in the case of FLY, where the basic bounds did not match for any query. It was also useful for LUBM, yielding matching bounds for 7 more queries.

- The refined treatment of existential rules described in Section 8.2, which yields the upper bound U_2, was especially effective for UOBM(1) and Reactome, where many existentially quantified rules were already satisfied by the lower bound materialisation.

- Finally, the refined treatment of disjunctive rules in Section 8.3, which yields the combined upper bound $U_{2|3}$, was instrumental in obtaining additional matching bounds for non-Horn ontologies. We could answer an additional 4 queries for UOBM(1), 31 for NPD, 5 for DBPedia, 13 for ChEMBL, 30 for Reactome, and 32 for Uniprot.

Overall, we obtained matching bounds for most queries in all our test ontologies: we could answer all queries for ChEMBL, and all but 1 for FLY and DBPedia, all but 2 for Reactome and LUBM(100), all but 4 for UOBM(1) and Uniprot, and all but 5 for NPD.

Subset Extraction Table 12.3 shows, for each dataset, the maximum percentage of facts and rules that are included in the relevant subset over all test queries with non-matching bounds. We can observe that subset extraction is effective in all cases in terms of both facts and rules. For Uniprot and DBPedia, the reduction in data

	LUBM	UOBM	Fly	NPD	DBPedia	Reactome	Uniprot
Facts	0.5%	10.4%	7.3%	16.5%	9×10^{-5}%	5.2%	4×10^{-4}%
Rules	3.7%	10.9%	0.9%	18.4%	2.4%	5.3%	1.1%

Table 12.3: Size of the largest subsets

	LUBM		UOBM				FLY
$L_2 + U_{2\|3}$	26	14	264	112	1470	264	344
+ Sum	26	14	264	0	1444	264	344
+ Dep	1	1	1	0	1	1	7

DBPedia	NPD	Reactome		UniProt
10	326	18	52	168
0	0	0	52	0
0	0	0	37	0

Table 12.4: ♯Hard calls to HermiT to fully answer each query

size was especially dramatic. It is also interesting to observe the large reduction in the number of rules for FLY, which is a rather complex ontology. Finally, subset extraction was least effective for NPD and UOBM, but even in these cases there was a reduction of almost one order of magnitude in the size of both ontology and dataset.

We now turn our attention to summarisation and dependency analysis. The effectiveness of these techniques was measured by the number of "hard" calls to HermiT that were required to fully answer each query, where a call to HermiT is considered hard if the knowledge base passed to HermiT is not a summary. The first row of Table 12.4 shows the number of gap answers for each query where the L_2 and $U_{2|3}$ bounds don't match. Without optimisation, we would have to call HermiT this number of times to fully answer each query. Row 2 (resp. row 3) shows the number of hard calls to HermiT after applying summarisation (resp. summarisation plus dependency analysis). As we mentioned above, there are respectively 5 and 4 queries with non-matching bounds for NPD and UniProt. However, for each of these groups, summarisation and dependency analysis have identical effects on all the queries in the group, so we present just one representative query for each ontology.

Summarisation As already discussed, summarisation enables PAGOdA to fully answer a number of test queries with non-empty gaps. It was instrumental in fully answering one query for each of UOBM(1), DBPedia and Reactome, as well as 5 queries for NPD, and 4 queries for Uniprot. Even in the cases where summarisation

did not suffice to fully answer the query, it was effective in reducing the size of the gap. For instance, for one of the queries for UOBM(1) we obtained 1,470 gap answers, of which 26 were ruled out by summarisation.

Dependency Analysis In LUBM(100) there were two queries with a gap of 26 answers and 14 answers, respectively; in both cases, all answers were merged into a single group, and hence a single call to HermiT sufficed to complete the computation. Similarly, in UOBM(1) a single call to HermiT was again sufficient, even though the three queries with a gap involved a large number of candidate answers. For FLY, there are 344 answers remaining to be verified after summarisation, but only 7 hard calls to HermiT were required. Finally, in the case of Reactome one query had 52 gap answers, but dependency analysis reduced the number of calls to HermiT to 37.

Chapter 13

Discussion

This thesis proposes a novel "pay-as-you-go" approach for conjunctive query answering that combines a datalog reasoner with a fully-fledged reasoner. The key feature of this approach is that it delegates the bulk of the computation to the scalable datalog reasoner and resorts to the expensive fully-fledged reasoner only as necessary to fully answer a query. The proposed techniques are very general and applicable to a wide range of reasoning problems. The main goal in practice, however, has been to realise this approach in a highly scalable and robust query answering system for OWL 2 ontologies, which is called PAGOdA. An extensive evaluation has not only confirmed the feasibility of this pay-as-you-go approach in practice, but also that the PAGOdA system significantly outperforms state-of-the art reasoning systems in terms of both robustness and scalability. In particular, the experiments using the ontologies in the EBI linked data platform have shown that PAGOdA is capable of fully answering queries over highly complex and expressive ontologies and realistic datasets containing hundreds of millions of facts.

Future Work

The work in this thesis opens up numerous exciting possibilities for future research.

Syntactical Conditions As we mentioned in Chapter 11, if an input ontology is captured by OWL 2 RL or $\mathcal{ELHO}^r_\bot$, the aggregated lower bound is tight for an arbitrary conjunctive query and hence PAGOdA can exploit RDFox or KARMA to fully answer queries over the input ontology. Since the aggregated lower bound is tight for both OWL 2 RL and $\mathcal{ELHO}^r_\bot$ ontologies, it could be tight for a more expressive ontology language that covers both. If there is a syntactical condition for such a target language, it can be easily incorporated into PAGOdA.

Similarly, it is possible that our upper bound is tight for certain ontology languages. We could also investigate the syntactical conditions of such languages. If an input ontology is captured by one of such languages, PAGOdA could fully answer queries over this ontology without resorting to a fully-fledged reasoner, even when the lower and upper bounds do not match.

Moreover, for a given ontology language, it might be possible to identify classes of conjunctive queries that can be fully answered by the lower or upper bound w.r.t. knowledge bases stemming from ontologies in the target language.

Fine-grained Existential Expansion For Horn knowledge bases, oblivious chase (Calì et al., 2013) is both sound and complete, although the termination of chase procedure is not guaranteed. However, there has been some work on syntactical conditions to ensure chase termination (Cuenca Grau et al., 2013). On the one hand, in some cases, we will probably get a tighter lower bound by introducing some new anonymous individuals for existentially quantified rules without affecting the termination of the procedure. If the chase termination is not guaranteed, we could ignore existential rules after a certain number of expansion steps. On the other hand, it is possible to expand our upper bound materialisation by introducing fresh individuals first and then reusing existing individuals. Consider an ontology $\{A \sqsubseteq \exists R.A\}$ and a dataset $\{A(a_1), A(a_2), A(a_3)\}$. The current approach for existentially quantified rules results in the lower and upper bound materialisation in Figure 13.1a and 13.1c, respectively. It is easy to see that the bounds computed w.r.t. the refined approach shown in Figure 13.1b and 13.1d turn out to be tighter. There are interesting topics for this fine-grained existential expansion, e.g. what is the best expanding depth of a specific input ontology, how to trade off between the size of the lower/upper bound materialisation and tightness of query answer, and if it is possible to do this existential expansion on demand.

Datalog Rewriting No rewriting exists from disjunctive datalog to datalog that preserves conjunctive query answers. A reason here is that the complexity of two problems is different. However, we could exploit sound or complete datalog rewriting for disjunctive datalog rules to obtain better lower and upper bound query answers. Moreover, it is also worth exploring rewriting techniques into datalog in the presence of both disjunctive rules and existentially quantified rules. Such rewriting techniques can be seamlessly incorporated into PAGOdA.

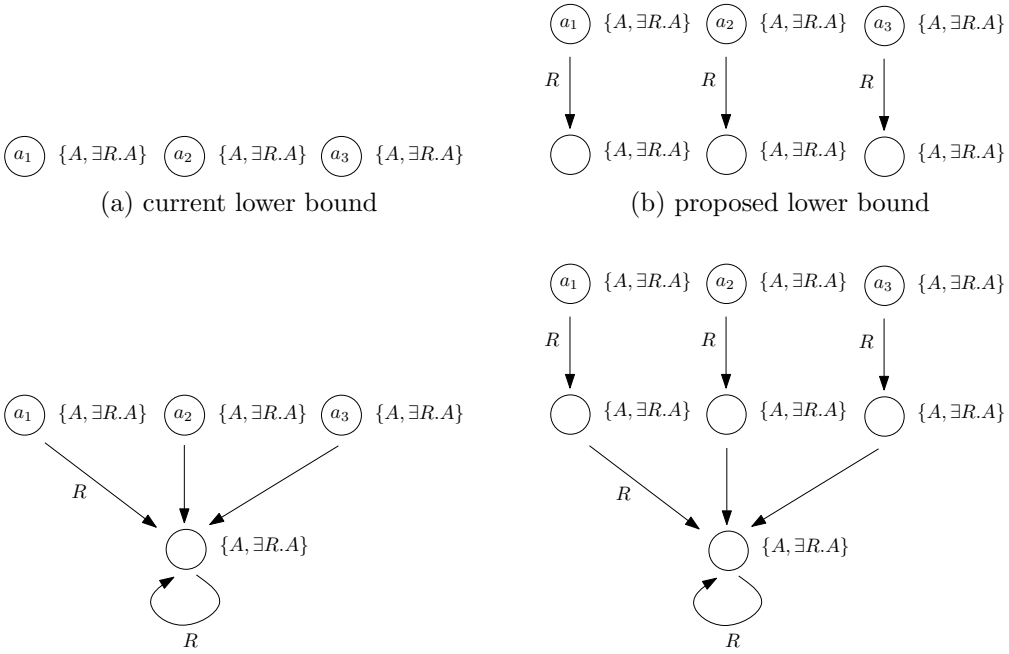


Figure 13.1: Existential expansion

Caching Query Answers This direction is very promising for applications like SemFacet (Arenas et al., 2014). Because a query composed by SemFacet is specified by a sequence of mouse clicks on facet values, two consecutive queries are closely related to each other, that is, the set of atoms of former one is typically a subset of the next one. Let q_1, q_2 be two consecutive queries. The certain answer to q_1 is also an upper bound to q_2. Potentially, this kind of information can be used to compute certain answer to q_2. Such techniques can be also exploited to optimise execution for queries with many atoms.

Query Answering Software If full OWL 2 reasoning is involved for an input query, it can be very costly to answer such a query. Otherwise, PAGOdA is able to compute query answers efficiently. It would be ideal to provide software to visualise of the query answering process inside PAGOdA, with an indication of how long it will take to finish. Users could then decide to either make do with incomplete answers or wait for complete answers.

Part IV

Appendix

Appendix A

The $\bot$-Related Optimisation

In Section 9.3.3, we described an optimisation to compute smaller relevant subsets of queries and gap answers without giving definitions and a correctness proof. In this chapter, we formalise this optimisation and prove the completeness of the computed relevant subsets. To this end, we fix an arbitrary normalised knowledge base $\mathcal{K}$, and an arbitrary conjunctive query q that might be the unsatisfiability query, and let N be the number of rules of the form ($\bot$-`Norm`) in $\mathcal{K}$.

To illustrate all aspects of this optimisation, we adopt the knowledge base and the query depicted in Figure A.1 as an running example.

Example A.1. Consider the knowledge base $\mathcal{K}_a$ in Figure A.1, the query $q_a(x) = B(x)$ and the set of answers $G_a = \{\{x \mapsto a\}\}$. The relevant subset for q_a and G_a computed by Definition 9.1 is $\mathcal{K}_a$ itself. However, it is not hard to check that r_7 and r_8 are not relevant the q since the query predicate B is not reachable from predicates G and H by any means. However, they are included in the relevant subset because they are relevant to derive $\bot_s$ from r_8. $\diamondsuit$

This example motivates us to look for more refined treatment for $\bot$-related subsets. For this purpose, instead of renaming $\bot$ in $\mathcal{K}$ by a single predicate $\bot_s$, we rename different occurrences of $\bot$ in $\mathcal{K}$ to a fresh nullary predicates $\bot_i$ without predefined meaning that is unique for each i and then process each $\bot_i$ separately.

Example A.2. Consider the knowledge base $\mathcal{K}_a$ in Figure A.1, there are four $\bot$s, each of which occurs in r_3, r_4, r_6, r_8.

$$\mathcal{K}_a^{\bot_1} = \{r_3, r_5, T(a,b), r_2, A(a)\}$$
$$\mathcal{K}_a^{\bot_2} = \{r_4, T(a,b), r_5\}$$
$$\mathcal{K}_a^{\bot_3} = \{r_6, r_1, A(a), r_2\}$$
$$\mathcal{K}_a^{\bot_4} = \{r_8, Q(a,c), r_7, A(a)\}$$

$r_1:\ A(x) \to B(x) \vee \exists y P(x,y)$
$r_2:\ A(x) \to C(x) \vee E(x)$
$r_3:\ S(x,y) \wedge C(x) \to \bot_{[1]}$
$r_4:\ F(x) \to \bot_{[2]}$
$r_5:\ T(x,y) \to S(x,y) \vee F(y)$
$r_6:\ P(x,y) \wedge E(x) \to \bot_{[3]}$
$r_7:\ A(x) \to G(x) \vee H(x)$
$r_8:\ Q(x,y) \wedge H(x) \to \bot_{[4]}$

$\Sigma_{\mathcal{K}_a} = \{r_1, \ldots, r_8\}$
$\mathcal{D}_{\mathcal{K}_a} = \{T(a,b),\ Q(a,c),\ A(a)\}$
$q_a(x) = B(x)$
$G_a = \{\{x \mapsto a\}\}$

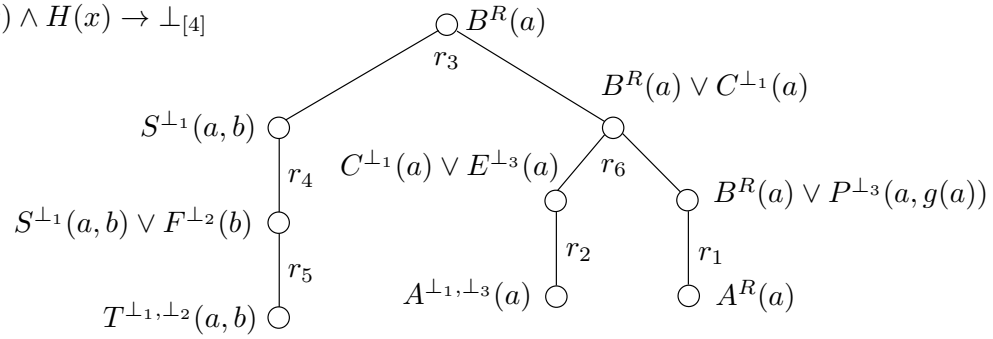


Figure A.1: Running example for the $\bot$-related optimisation

It is not hard to observe that $\mathcal{K}^{\bot_4}$ is not relevant to any proof of $q(a)$, since predicates G and H could never lead to B. ♢

Next we formalise the renaming of $\bot$ in a knowledge base.

Definition A.3. Let $\{r_1, \ldots, r_N\}$ be the subset of rules of the form ($\bot$-Norm) in $\Sigma_{\mathcal{K}}$, let $\bot_1, \ldots, \bot_N$ be N different fresh predicates. The $\bot$-*extension* $\mathsf{ext}(\mathcal{K})$ of $\mathcal{K}$ is defined as the smallest knowledge base containing all rules and facts from

$$\mathcal{K} \setminus \{r_1, \ldots, r_N\} \cup \{\mathsf{body}(r_i) \to \bot_i,\ \bot_i \to \bot \mid 1 \leq i \leq N\}.$$

For each rule $r \in \mathcal{K}$ of the form ($\bot$-Norm), the $\bot$-index of r, written $N_\bot(r)$, is defined as the number i such that $\mathsf{body}(r) \to \bot_i$ is a rule in $\mathsf{ext}(\mathcal{K})$. ♢

It is obvious that $\mathsf{ext}(\mathcal{K})$ is a conservative extension of $\mathcal{K}$ and $\mathsf{ext}(\mathcal{K}) \models \mathcal{K}$. The relevant subset of $\bot_i$ in $\mathcal{K}$ are formally defined next.

Definition A.4. Consider the knowledge base $\mathcal{K}$ and the number N, let b_i be a fresh unique constant for each $1 \leq i \leq N$, let Rel_i be a fresh unary predicate for each $1 \leq i \leq N$, and let d_r be a fresh unique constant to each r in $\sigma_{\mathcal{K}}$. Moreover, for each predicate P in $\mathsf{ext}(\mathcal{K})$, let $P^{\bot_1}, \ldots, P^{\bot_N}, P^{\bot_A}$ be fresh predicates of the same arity as P. Furthermore, let $\mathsf{overlap}$ be a fresh transitive and symmetric relation. For each $1 \leq i \leq N$ and a rule $r \in \mathcal{K}$, let $\mathsf{move}_i(r)$ be the following set of conjunction of atoms:

- $\{\bot^{\bot_i}_{N_\bot(r)}\}$ if r of the form ($\bot$-Norm);
- $\{\gamma_1^{\bot_i}(\mathbf{x}, \mathbf{z}_1)\}$ if r is of the form ($\exists$-Norm); and
- $\{\gamma_1^{\bot_A}(\mathbf{x}) \wedge \cdots \wedge \gamma_{j-1}^{\bot_A}(\mathbf{x}) \wedge \gamma_j^{\bot_i}(\mathbf{x}) \wedge \gamma_{j+1}^{\bot_A}(\mathbf{x}) \wedge \cdots \wedge \gamma_m^{\bot_A}(\mathbf{x}) \mid 1 \leq j \leq m\}$ if r is of the form ($\vee$-Norm).

Then, $\Delta_i(r)$ is the set of following rules:

- $\chi \wedge \mathsf{body}(r) \to \mathsf{Rel}_i(\mathsf{d}_r)$ for each $\chi \in \mathsf{move}_i(r)$,
- $\chi \wedge \mathsf{body}(r) \to \beta_k^{\bot_i}$ for each atom β_k in $\mathsf{body}(r)$ and $\chi \in \mathsf{move}_i(r)$, and
- $\bigwedge_{t \neq j,k} \gamma_t^{\bot_A}(\mathbf{x}) \wedge \gamma_j^{\bot_i}(\mathbf{x}) \wedge \gamma_k^{\bot_\ell}(\mathbf{x}) \wedge \mathsf{body}(r) \to \mathsf{overlap}(b_i, b_\ell)$ for any $1 \leq j < k \leq m$ and $1 \leq \ell \leq N$.

The $\bot$-tracking knowledge base $\mathsf{track}_\bot(\mathcal{K})$ is the smallest knowledge base containing

(i) all facts in the c-chase of $\mathcal{K}$;

(ii) all rules in $\Delta_i(r)$ for each $1 \leq i \leq N$ and each rule r in $\mathcal{K}$;

(iii) a fact $\bot_i^{\bot_i}$ for each $1 \leq i \leq N$;

(iv) $\mathsf{overlap}(x, y) \to \mathsf{overlap}(y, x)$, $\mathsf{overlap}(x, y) \wedge \mathsf{overlap}(y, z) \to \mathsf{overlap}(x, z)$; and

(v) a rule $P^{\bot_i}(\mathbf{x}) \to P^{\bot_A}(\mathbf{x})$ for each $1 \leq i \leq N$ and each predicate P in $\mathsf{ext}(\mathcal{K})$.

For each $1 \leq i \leq N$, $\mathcal{K}^{\bot_i}$ is the smallest knowledge base with each $r \in \Sigma_\mathcal{K}$ such that $\mathsf{track}_\bot(\mathcal{K}) \models \mathsf{Rel}_i(\mathsf{d}_r)$ and each $\alpha \in \mathcal{D}_\mathcal{K}$ such that $\mathsf{track}_\bot(\mathcal{K}) \models \alpha^{\bot_i}$. ♢

Now we are ready to define relevant subsets for conjunctive queries including the unsatisfiability query.

Definition A.5. Let N, b_i, $P^{\bot_i}$, $P^{\bot_A}$, $\mathsf{overlap}$, d_r, and $\mathsf{track}_\bot(\mathcal{K})$ be as required in Definition A.4. Let G be a set of possible answers to q, let b_q be a fresh constant for query q, and let Rel be a fresh unary predicate. Furthermore, for each predicate P in $\mathcal{K} \cup \mathcal{R}_q$, let P^R, P^{R_A} be fresh predicates of the same arity as P. For a rule $r \in \mathcal{K} \cup \mathcal{R}_q$, let $\mathsf{move}(r)$ be the following set of conjunction of atoms:

- $\emptyset$ if r is of the form ($\bot$-Norm);
- $\{\gamma_1^R(\mathbf{x}, \mathbf{z}_1)\}$ if r is of the form ($\exists$-Norm); and

- $\{\gamma_1^{R_A}(\mathbf{x}) \wedge \cdots \wedge \gamma_{j-1}^{R_A}(\mathbf{x}) \wedge \gamma_j^{R}(\mathbf{x}) \wedge \gamma_{j+1}^{R_A}(\mathbf{x}) \wedge \cdots \wedge \gamma_m^{R_A}(\mathbf{x}) \mid 1 \leq j \leq m\}$ if r if of the form ($\vee$-Norm).

Then, $\Delta(r)$ is the set of following rules:

- $\chi \wedge \mathsf{body}(r) \rightarrow \mathsf{Rel}_i(\mathsf{d}_r)$ for each $\chi \in \mathsf{move}(r)$,
- $\chi \wedge \mathsf{body}(r) \rightarrow \beta_i^R$ for each atom β_i in $\mathsf{body}(r)$ and $\chi \in \mathsf{move}(r)$, and
- if r is of the form ($\perp$-Norm),

$$P_\perp^R \wedge \mathsf{body}(r) \rightarrow \mathsf{overlap}(b_q, b_{N_\perp(r)});$$

otherwise, for any $1 \leq i, j \leq m$, $i \neq j$ and $1 \leq \ell \leq N$,

$$\bigwedge_{k \neq i,j} \gamma_k^{R_A}(\mathbf{x}) \wedge \gamma_i^{R}(\mathbf{x}) \wedge \gamma_j^{\perp_\ell}(\mathbf{x}) \wedge \mathsf{body}(r) \rightarrow \mathsf{overlap}(b_q, b_\ell).$$

The tracking knowledge base $\mathsf{track}(\mathcal{K}, q, G)$ is the smallest knowledge base containing

(i) all facts in the materialisation of $\mathsf{track}_\perp(\mathcal{K})$;

(ii) all rules in $\Delta(r)$ for each rule r in $\mathcal{K} \cup \mathcal{R}(q)$;

(iii) a fact $P_q^R(\mathbf{x})\sigma$ for each $\sigma \in G$;

(iv) $\mathsf{overlap}(x, y) \rightarrow \mathsf{overlap}(y, x)$, $\mathsf{overlap}(x, y) \wedge \mathsf{overlap}(y, z) \rightarrow \mathsf{overlap}(x, z)$; and

(v) rules $P^{\perp_A}(\mathbf{x}) \rightarrow P^{R_A}(\mathbf{x})$ and $P^R(\mathbf{x}) \rightarrow P^{R_A}(\mathbf{x})$ for each predicate P in $\mathsf{ext}(\mathcal{K})$,

The subset of $\mathcal{K}$ relevant to q and G, denoted $\mathcal{K}^{q,G}$, is the smallest knowledge base containing

- all rules and facts in $\mathcal{K}^{\perp_i}$ for each $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_i)$;
- rules $r \in \Sigma_\mathcal{K}$ such that $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{Rel}(\mathsf{d}_r)$; and
- facts $\alpha \in \mathcal{D}_\mathcal{K}$ such that $\mathsf{track}(\mathcal{K}, q, G) \models \alpha^R$. $\diamondsuit$

The properties in Theorem 9.4 still hold under our new definition of tracking knowledge base.

Theorem A.6. *Under Definition A.5, the following properties hold:*

1. Assume that $L^\perp = \emptyset$. Then, $\mathcal{K}$ is unsatisfiable iff $\mathcal{K}^{\perp,U^\perp}$ is unsatisfiable.

2. *Let q be different from $\bot$ and let G be any non-empty set of possible answers to q w.r.t. $\mathcal{K}$. If $\mathcal{K}$ is satisfiable, then $\sigma \in \mathsf{cert}(q, \mathcal{K})$ iff $\sigma \in \mathsf{cert}(q, \mathcal{K}^{q,G})$ for every $\sigma \in G$.*

The following lemma is the key to prove Theorem 9.4. We first give the lemma, and then show Theorem 9.4 follows from the lemma.

Lemma A.7. *Let $\rho = (T, \lambda)$ be a hyperresolution derivation from $\mathcal{K} \cup \mathcal{R}_q$ of a query goal from $\Upsilon(q, G)$, then $\mathsf{support}(\rho) \subseteq \mathcal{K}^{q,G}$.*

Proof of Theorem A.6. The "if" direction of Theorem A.6.1 and Theorem A.6.2 follows directly from the monotonicity of first-order logic. To show the "only if" direction of Theorem A.6.1, assume that $\mathcal{K}$ is unsatisfiable. By Theorem 8.11, Theorem 8.16 and Definition 8.19, we have $U^{\bot} \neq \emptyset$ and thus $G^{\bot} \neq \emptyset$. Then there exists a hyperresolution derivation ρ_1 of $\Box$ from $\mathcal{K}$. Since $\Upsilon(\bot, G^{\bot}) = \{\Box\}$, we know that $\mathsf{support}(\rho_1) \subseteq \mathcal{K}^{\bot}$ by Lemma A.7. So $\mathcal{K}^{\bot}$ is unsatisfiable. To show the "only if" direction of Theorem A.6.2, assume that $\sigma \in G$ and $\sigma \in \mathsf{cert}(q, \mathcal{K})$. Then there exists a hyperresolution ρ_2 of $P_q(\mathbf{x})\sigma$ from $\mathcal{K} \cup \mathcal{R}_q$. Similarly, by Lemma A.7, we know that $\mathsf{support}(\rho_2) \subseteq \mathcal{K}^{q,G}$ and hence $\sigma \in \mathsf{cert}(q, \mathcal{K}^{q,G})$. □

From now on, we fix an arbitrary G and a hyperresolution derivation $\rho = (T, \lambda)$ of $\alpha \in \Upsilon(q, G)$ from $\mathcal{K} \cup \mathcal{R}_q$. Let $\mathcal{H} = \mathsf{split}(\mathcal{K})$. We have already established (see the proof of Theorem 8.11) that $\mathsf{c\text{-}Chase}_{\mathcal{K}}$ is a model of $\mathcal{H}$. By Lemma 2.5, there exists a constant-preserving homomorphism τ from $\mathsf{skChase}(\mathcal{H})$ to $\mathsf{c\text{-}Chase}_{\mathcal{K}}$. For convenience, we will use Proposition 2.4 implicitly in the rest of this chapter.

Example A.8. Consider the example depicted in Figure A.1, the relevant fragment for q_a and G_a is now $\mathcal{K}_a \setminus \{Q(a, c), r_7, r_8\}$ using the refined treatment for $\bot$. ♢

Proposition A.9. *If $\mathsf{track}(\mathcal{K}, q, G) \models \alpha^{\bot_\ell}$, then $\mathsf{track}_{\bot}(\mathcal{K}) \models \alpha^{\bot_\ell}$.*

The proposition is trivial as predicates of the form $P^{\bot_\ell}$ never appear in the head of any rules in $\mathsf{track}(\mathcal{K}, q, G)$.

Definition A.10. Symbols of the form R or $\bot_i$ for any $1 \leq i \leq \ell$ are called *superscripts*. We next inductively define a series of *superscripts functions* δ such that δ_u for each node u in T that maps each atom in $\lambda(u)$ to a set of superscripts.

- For the root u of T, $\delta_u(\alpha) = \{R\}$ for each atom α in $\lambda(u)$.

- Consider a node u in T that δ_u is defined. We define superscript functions for its children $v_1, \ldots, v_n$ accordingly. Let r be the rule in $\mathcal{K} \cup \mathcal{R}_q$ such that $\mathsf{sk}(r)$ is the main premise in the hyperresolution step to obtain $\lambda(u)$ with MGU σ, i.e. $\lambda(u) = \gamma_1\sigma \vee \cdots \vee \gamma_m\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is the hyperresolvent of $\lambda(v_i) = \beta_i\sigma \vee \chi_i$ for $1 \leq i \leq n$ and $\mathsf{sk}(r) = \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \gamma_1 \vee \cdots \vee \gamma_m$.

$$\delta_{v_i}(\beta_i\sigma) = \begin{cases} \{\bot_{N_\bot(r)}\} & \text{if } m = 0; \\ \{R\} & \text{if } \delta_u(\gamma_j\sigma) = \{R\} \text{ for any } 1 \leq j \leq m; \\ \bigcup_{j=1}^{m} \delta_u(\gamma_j\sigma) & \text{otherwise;} \end{cases}$$

$$\delta_{v_i}(\alpha) = \delta_u(\alpha) \qquad \text{if } \alpha \text{ is an atom in } \chi_i.$$

Furthermore, the result of applying δ_u to a ground clause is defined as the union of the result applying δ_u to each atom in it, i.e. $\delta_u(\alpha_1 \vee \cdots \vee \alpha_k) = \bigcup_{i=1}^{k} \delta_u(\alpha_i)$. ♢

Example A.11. Consider the hyperresolution derivation depicted in Figure A.1. The superscripts function values of it are also presented in the graph as superscripts in on the predicate of each atoms. The intuition behind is if $\bot_i$ is in the superscripts of an atom α, then the atom α is involved in a process to derive $\bot_i$, otherwise, α is involved to derive a query answer. For instance, the left leaf $A(a)$ is used to derive $C(a) \vee E(a)$, which are eliminated in the latter steps by $S(x, y) \wedge C(x) \to \bot$ and $P(x, y) \wedge E(x) \to \bot$ respectively. The right leaf $A(a)$ has the superscript R and the superscript is preserved until the root node. ♢

By Lemma 8.5 and that τ is a constant-preserving homomorphism from $\mathsf{skChase}(\mathcal{H})$ to $\mathsf{c\text{-}Chase}_\mathcal{K}$, we then have the following property.

Proposition A.12. *For each resolution step in ρ, let σ be the MGU in this step, and let r be the rule in $\mathcal{K} \cup \mathcal{R}_q$ such that $\mathsf{sk}(r) = \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \gamma_1 \vee \cdots \vee \gamma_m$ is the main premise in this step. We have*

$$\beta_i\sigma\tau \in \mathsf{track}_\bot(\mathcal{K}) \text{ for } 1 \leq i \leq n.$$

Lemma A.13. *For each node u in T, each atom α occurs in $\lambda(v)$ and each superscript, $\mathsf{S} \in \delta_v(\alpha)$ $\mathsf{track}(\mathcal{K}, q, G) \models (\alpha\tau)^\mathsf{S}$.*

Proof. We proceed by induction top-down on the structure of T.

Base case: In the base case we have that v is the root of T.

- If $\lambda(v) = \square$, then this claim holds vacuously.

- If $\lambda(v) = P_q(\mathbf{x})\sigma$, then by the definition of $\mathsf{track}(\mathcal{K}, q, G)$ (Point *iii*) we have that $P_q^R(\mathbf{x})\sigma \in \mathsf{track}(\mathcal{K}, q, G)$ and hence the property holds.

Inductive case: For the inductive step, assume that the property hold for a node u. We next show that they also hold for the children $v_1, \ldots, v_n$ of u. Let r be the rule in $\mathcal{K} \cup \mathcal{R}_q$ such that $\mathsf{sk}(r)$ is the main premise in the hyperresolution step to obtain $\lambda(u)$ with MGU σ, i.e. $\lambda(u) = \gamma_1\sigma \vee \cdots \vee \gamma_m\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is the hyperresolvent of $\lambda(v_i) = \beta_i\sigma \vee \chi_i$ for $1 \leq i \leq n$ and $\mathsf{sk}(r) = \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \gamma_1 \vee \cdots \vee \gamma_m$. By induction hypothesis, we have $\mathsf{track}(\mathcal{K}, q, G) \models (\alpha\tau)^{\mathsf{S}}$ for each α in χ_i and each $\mathsf{S} \in \lambda_{v_i}(\alpha)$. Therefore, it suffices to show that $\mathsf{track}(\mathcal{K}, q, G) \models (\beta_i\sigma\tau)^{\mathsf{S}}$ for each i and each $\mathsf{S} \in \lambda_{v_i}(\beta_i\sigma)$. To this end, we distinguish three cases by the structure of r.

- If $m = 0$, then for each $1 \leq i \leq n$ we have $\delta_{v_i}(\beta_i\sigma) = \{\bot_\ell\}$ with $\ell = N_\bot(r)$. By Proposition A.12, the fact $\bot_\ell^{\bot_\ell} \in \mathsf{track}_\bot(\mathcal{K})$ and the rule $\bot_\ell^{\bot_\ell} \wedge \mathsf{body}(r) \to \beta_i^{\bot_\ell}$ are in $\mathsf{track}_\bot(\mathcal{K})$, we have $\mathsf{track}_\bot(\mathcal{K}) \models (\beta_i\sigma\tau)^{\bot_\ell}$ and hence $\mathsf{track}(\mathcal{K}, q, G) \models (\beta_i\sigma\tau)^{\bot_\ell}$ as facts.

- If $\delta_u(\gamma_j\sigma) = \{R\}$ holds for some $1 \leq j \leq m$. Then $\delta_{v_i}(\beta_i\sigma) = \{R\}$ for each $1 \leq i \leq n$. By the induction hypothesis, we have that $\mathsf{track}(\mathcal{K}, q, G) \models (\gamma_k\sigma\tau)^{\mathsf{S}}$ for each $1 \leq k \leq m$ and each $\mathsf{S} \in \delta_u(\gamma_k\sigma)$. Then $\mathsf{track}(\mathcal{K}, q, G) \models (\gamma_k\sigma\tau)^{R_A}$, since rules of the form $P^R(\mathbf{x}) \to P^{R_A}(\mathbf{x})$, $P^{\bot_A}(\mathbf{x}) \to P^{R_A}(\mathbf{x})$ and $P^{\bot_\ell}(\mathbf{x}) \to P^{\bot_A}(\mathbf{x})$ are satisfied by the materialisation of $\mathsf{track}(\mathcal{K}, q, G)$. Because Proposition A.12 and that $\gamma_1^{R_A} \wedge \cdots \wedge \gamma_{j-1}^{R_A} \wedge \gamma_j^R \wedge \gamma_{j+1}^{R_A} \wedge \cdots \wedge \gamma_m^{R_A} \wedge \mathsf{body}(r) \to \beta_i^R$ is a rule in $\mathsf{track}(\mathcal{K}, q, G)$ for each $1 \leq i \leq n$, we have $\mathsf{track}(\mathcal{K}, q, G) \models (\beta_i\sigma\tau)^R$.

- Otherwise, we have $\delta_{v_i}(\beta_i\sigma) = \bigcup_{j=1}^m \delta_u(\gamma_j\sigma)$. For each $1 \leq i \leq m$, we have $\emptyset \subset \delta_u(\gamma_i\sigma) \subseteq \{\bot_1, \ldots, \bot_N\}$. By the induction hypothesis, for each $\mathsf{S} \in \delta_u(\gamma_i\sigma)$, we have $\mathsf{track}(\mathcal{K}, q, G) \models (\gamma_i\sigma\tau)^{\mathsf{S}}$ and then $\mathsf{track}_\bot(\mathcal{K}) \models (\gamma_i\sigma\tau)^{\mathsf{S}}$ by Proposition A.9. Because all rules of the form $P^{\bot_\ell}(\mathbf{x}) \to P^{\bot_A}(\mathbf{x})$ are in $\mathsf{track}_\bot(\mathcal{K})$, $\mathsf{track}_\bot(\mathcal{K}) \models (\gamma_i\sigma\tau)^{\bot_A}$ for each i. For each $\mathsf{S} \in \delta_{v_i}(\beta_i\sigma)$, by Proposition A.12 and that $\gamma_1^{\bot_A} \wedge \cdots \wedge \gamma_{j-1}^{\bot_A} \wedge \gamma_j^{\mathsf{S}} \wedge \gamma_{j+1}^{\bot_A} \wedge \cdots \wedge \gamma_m^{\bot_A} \wedge \mathsf{body}(r) \to \beta_i^{\mathsf{S}}$ is a rule in $\mathsf{track}_\bot(\mathcal{K})$, we have $\mathsf{track}_\bot(\mathcal{K}) \models (\beta_i\sigma\tau)^{\mathsf{S}}$. □

Lemma A.14. *For each node w in T, if $\{\bot_\ell, \bot_{\ell'}\} \subseteq \delta_w(\alpha)$ for an atom α in $\lambda(w)$, then* $\mathsf{track}_\bot(\mathcal{K}) \models \mathsf{overlap}(b_\ell, b_{\ell'})$.

Proof. Let $\bot_\ell, \bot_{\ell'}$ be two superscripts that satisfies the condition in this claim. Let $u_1, \ldots, u_L$ be the sequence from w to the root of T, and let k be the smallest number

that $1 < k \leq L$, $\{\bot_\ell, \bot_{\ell'}\} \subseteq \delta_{u_{k-1}}(\alpha)$ for an atom α in $\lambda(u_{k-1})$, but $\{\bot_\ell, \bot_{\ell'}\} \not\subseteq \delta_{u_k}(\alpha')$ for any atom α' in $\lambda(u_k)$. The existence of k is guaranteed by facts that $\{\bot_\ell, \bot_{\ell'}\} \subseteq \delta_{u_1}(\alpha)$ for an atom α in $\lambda(u_1)$ and $\{\bot_\ell, \bot_{\ell'}\} \not\subseteq \delta_{u_L}(\alpha')$ for any atom α' in $\lambda(u_L)$. Let u be such a node u_k, let $v_1, \ldots, v_n$ be the children of u, let r be the rule in $\mathcal{K} \cup \mathcal{R}_q$ such that $\mathsf{sk}(r)$ is the main premise in the hyperresolution step to obtain $\lambda(u)$ with MGU σ, i.e. $\lambda(u) = \gamma_1\sigma \vee \cdots \vee \gamma_m\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is the hyperresolvent of $\lambda(v_i) = \beta_i\sigma \vee \chi_i$ for $1 \leq i \leq n$ and $\mathsf{sk}(r) = \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \gamma_1 \vee \cdots \vee \gamma_m$.

- If $\ell = \ell'$, we have $\bot_\ell \in \delta_{v_i}(\lambda(v_i))$ but $\bot_\ell \notin \delta_u(\lambda(u))$. Then r is of the form ($\bot$-`Norm`) that $N_\bot(r) = \ell$. Therefore, $\bot_\ell^{\bot_\ell} \wedge \mathsf{body}(r) \to \mathsf{overlap}(b_\ell, b_\ell)$ is a rule in $\mathsf{track}_\bot(\mathcal{K})$. By Proposition A.12 and $\bot_\ell^{\bot_\ell} \in \mathsf{track}_\bot(\mathcal{K})$, we have $\mathsf{track}_\bot(\mathcal{K}) \models \mathsf{overlap}(b_\ell, b_\ell)$

- Otherwise r can only be a disjunctive rule with $m > 1$, $\{\bot_\ell, \bot_{\ell'}\} \subseteq \bigcup_{i=1}^m \delta_u(\gamma_i\sigma)$ and $R \notin \bigcup_{i=1}^m \delta_u(\gamma_i\sigma)$. Thus, $\emptyset \subset \delta_u(\gamma_i\sigma) \subseteq \{\bot_1, \ldots, \bot_N\}$ for each i. Let k, k' be two indexes such that $\bot_\ell \in \delta_u(\gamma_k\sigma)$ and $\bot_{\ell'} \in \delta_u(\gamma_{k'}\sigma)$. Thus, we have $k \neq k'$ since there is no α' in $\lambda(u)$ such that $\{\bot_\ell, \bot_{\ell'}\} \subseteq \delta_u(\alpha')$. W.l.o.g. we can assume that $k < k'$. By Lemma A.13 we have $\mathsf{track}(\mathcal{K}, q, G) \models (\gamma_i\sigma\tau)^{\bot_{j_i}}$ for some j_i. Since rules of the form $P^{\bot_{j_i}}(\mathbf{x}) \to P^{\bot_A}(\mathbf{x})$ are included in $\mathsf{track}_\bot(\mathcal{K})$, we have $\mathsf{track}(\mathcal{K}, q, G) \models (\gamma_i\sigma\tau)^{\bot_A}$. So $\mathsf{track}_\bot(\mathcal{K}) \models (\gamma_i\sigma\tau)^{\bot_A}$. Since $\bigwedge_{i \neq k,k'} \gamma_i^{\bot_A} \wedge \gamma_k^{\bot_\ell} \wedge \gamma_{k'}^{\bot_{\ell'}} \wedge \mathsf{body}(r) \to \mathsf{overlap}(b_\ell, b_{\ell'})$ is a rule in $\mathsf{track}_\bot(\mathcal{K})$ and Proposition A.12 holds, we know that $\mathsf{track}_\bot(\mathcal{K}) \models \mathsf{overlap}(b_\ell, b_{\ell'})$. □

Example A.15. Consider the derivation in Figure A.1. $\bot_1, \bot_2$ both appear in the superscripts of $T(a, b)$. So u_1 is the leftmost node $T(a, b)$. So $\{u_1, \ldots, u_4\}$ in the proof corresponds to

$$\{T(a, b), S(a, b) \vee F(b), S(a, b), B(a)\}$$

in the figure. Then $u = u_2$, and $\bot_1, \bot_2$ are merged in a resolution step with the main premise $\mathsf{sk}(r_5)$. Since

$$S^{\bot_1}(x, y) \wedge F^{\bot_2} \wedge T(x, y) \to \mathsf{overlap}(b_1, b_2)$$

is a rule in $\mathsf{track}_\bot(\mathcal{K}_a)$, $\mathsf{overlap}(b_1, b_2)$ is derived in $\mathsf{track}_\bot(\mathcal{K}_a)$. ♢

Lemma A.16. *For each node u in T, if $R \notin \delta_u(\lambda(u))$ and $\{\bot_\ell, \bot_{\ell'}\} \subseteq \delta_u(\lambda(u))$, then we have $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_\ell, b_{\ell'})$.*

Proof. We prove this lemma by bottom-up induction on the structure of T.

Base case: If u is a leaf node, then $\lambda(u)$ is a single atom and this claim is entailed by Lemma A.14.

Inductive case: Let $v_1, \ldots, v_n$ be the children of u, and let r be the rule in $\mathcal{K} \cup \mathcal{R}_q$ such that $\mathsf{sk}(r)$ is the main premise in the hyperresolution step to obtain $\lambda(u)$ with MGU σ, i.e. $\lambda(u) = \psi\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is the hyperresolvent of $\lambda(v_i) = \beta_i\sigma \vee \chi_i$ for $1 \leq i \leq n$ and $\mathsf{sk}(r) = \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \psi$.

- If r is of the form ($\bot$-`Norm`), $\lambda(u) = \chi_1 \vee \cdots \vee \chi_n$ and $\delta_{v_i}(\alpha) = \delta_u(\alpha)$ for each $\alpha \in \chi_i$ and $\delta_{v_i}(\beta_i\sigma) = \{\bot_{N_\bot(r)}\}$. By the induction hypothesis, we have that the claim holds for each child v_i. So $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_{N(r)}, b_\ell)$ for each $\bot_\ell \in \delta_u(\chi_i)$ and each i. Since $\mathsf{overlap}$ is both transitive and symmetric, then we have the claim holds for u.

- Otherwise, since $R \notin \delta_u(\lambda(u))$, ℓ exists such that $\bot_\ell \in \delta_u(\psi\sigma)$. Then $\bot_\ell \in \delta_{v_i}(\beta_i\sigma)$ for each i. By the induction hypothesis, we have that the claim holds for each child v_i. So $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_\ell, b_{\ell'})$ for each $\bot_{\ell'} \in \delta_u(\chi_i)$ and each i. Since $\mathsf{overlap}$ is both transitive and symmetric, then we have the claim holds for u. □

The intuition behind the above proof is that if R is not in the superscripts of a node, all the superscripts of the node will be preserved to its children, and two superscripts can only be merged by disjunctive rules.

Example A.17. Consider the derivation in Figure A.1. $\bot_1, \bot_2$ both occur in the superscripts of $S(a, b) \vee F(b)$; $\bot_1, \bot_3$ occur in the superscripts of $C(a) \vee E(a)$. For both cases, the superscripts are merged in the next step and hence they are in the superscripts of a leaf node. ♢

Lemma A.18. *For each node u in T, if $\{R, \bot_\ell\} \subseteq \delta_u(\lambda(u))$, we have $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_\ell)$.*

Proof. We prove this lemma by bottom-up induction on the structure of T.

Base case: If v is a leaf in T, then $\delta_u(\lambda(u)) = \{R\}$ or $\emptyset \subset \delta_u(\lambda(u)) \subseteq \{\bot_1, \ldots, \bot_N\}$. So this claim holds vacuously.

Inductive case: If $\{R, \bot_\ell\} \subseteq \delta_u(\lambda(u))$, let $v_1, \ldots, v_n$ be the children of u and let r be the rule in $\mathcal{K} \cup \mathcal{R}_q$ such that $\mathsf{sk}(r)$ is the main premise in the hyperresolution step to

obtain $\lambda(u)$ with MGU σ, i.e. $\lambda(u) = \gamma_1\sigma \vee \cdots \vee \gamma_m\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is the hyperresolvent of $\lambda(v_i) = \beta_i\sigma \vee \chi_i$ for $1 \leq i \leq n$ and $\mathsf{sk}(r) = \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \gamma_1 \vee \cdots \vee \gamma_m$. If $\{R, \bot_\ell\} \subseteq \delta_v(\lambda(v))$ for a children v, then this claim holds by induction hypothesis. So we next assume that there is no such child.

- If r is of the form ($\bot$-Norm), $\bot_{N_\bot(r)} \in \delta_{v_i}(\beta_i\sigma)$ for each i. Then there are two children v and v' of u such that $R \in \delta_v(\lambda(v))$ and $\bot_\ell \in \delta_{v'}(\lambda(v'))$. By induction hypothesis, we have $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_{N_\bot(r)})$ and $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_\ell, b_{N_\bot(r)})$. Since $\mathsf{overlap}$ is both transitive and symmetric, we have $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_l)$.

- Otherwise, we distinguish two cases.

  - If there exists k such that $\delta_u(\gamma_k\sigma) = \{R\}$, then $\delta_{v_i}(\beta_i\sigma) = \{R\}$. If $\bot_\ell \in \delta_u(\chi_i)$ for some i, then the claim holds due to the induction hypothesis. So we assume that k' exists such that $\bot_\ell \in \delta_u(\gamma_{k'}\sigma)$. By Definition A.10 we have $k \neq k'$; by Lemma A.13 we have $\mathsf{track}(\mathcal{K}, q, G) \models \{(\gamma_{k'}\sigma\tau)^{\bot_\ell}, (\gamma_k\sigma\tau)^R\}$. Due to Proposition A.12 holds and that $\bigwedge_{i\neq k,k'} \gamma_i^{R_A} \wedge \gamma_k^R \wedge \gamma_{k'}^{\bot_\ell} \wedge \mathsf{body}(r) \to \mathsf{overlap}(b_q, b_\ell)$ is in $\mathsf{track}(\mathcal{K}, q, G)$, we have $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_\ell)$.
  - Otherwise, $\emptyset \subset \bigvee_{i=1}^m \delta_u(\gamma_i\sigma) \subseteq \{\bot_1, \ldots, \bot_N\}$. So there exists two children v_k and $v_{k'}$ such that $R \in \delta_{v_k}(\lambda(v_k))$ and $\bot_\ell \in \delta_{v_{k'}}(\lambda(v_{k'}))$. We can further assume that $k \neq k'$ and $R \notin \delta_{v_{k'}}(\lambda(v_{k'}))$, because otherwise the claim is entailed by the induction hypothesis. Since ℓ' exists such that $\bot_{\ell'} \in \bigvee_{i=1}^m \delta_u(\gamma_i\sigma)$, then $\bot_{\ell'} \in \delta_{v_k}(\lambda(v_k))$ and $\bot_{\ell'} \in \delta_{v_{k'}}(\lambda(v_{k'}))$. By induction hypothesis, we have $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_{\ell'})$. Moreover, by Lemma A.16, we have $\mathsf{track}_\bot(\mathcal{K}) \models \mathsf{overlap}(b_\ell, b_{\ell'})$. Since $\mathsf{overlap}$ is transitive and symmetric, we have $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_\ell)$. □

Example A.19. Consider the node $B(a) \vee C(a)$ in Figure A.1, the superscripts of it is $\{R, \bot_1\}$. In the hyperresolution step to obtain it, the superscripts are splitted into $\{\bot_1, \bot_3\}, \{R, \bot_3\}$ to its children. Then if $\mathsf{overlap}(b_1, b_3)$ and $\mathsf{overlap}(b_q, b_3)$, then $\mathsf{overlap}(b_q, b_1)$ holds. So we can prove it inductively. ♢

Lemma A.20. *For each node u in T, if $\bot_\ell \in \delta_u(\lambda(u))$, then* $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_\ell)$.

Proof. We proceed by top-down induction on the structure of T.

Base case: The claim holds vacuously for the root node of T.

Inductive case: Let u be a node for which the claim holds. We next prove that the claim holds for all its children $v_1, \ldots, v_n$. Let r be the rule in $\mathcal{K} \cup \mathcal{R}_q$ such that $\mathsf{sk}(r)$ is the main premise in the hyperresolution step to obtain $\lambda(u)$ with MGU σ, i.e. $\lambda(u) = \gamma_1\sigma \vee \cdots \vee \gamma_m\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is the hyperresolvent of $\lambda(v_i) = \beta_i\sigma \vee \chi_i$ for $1 \leq i \leq n$ and $\mathsf{sk}(r) = \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \gamma_1 \vee \cdots \vee \gamma_m$. If r is not of the form ($\bot$-`Norm`), then $\bigcup_{i=1}^{n} \delta_{v_i}(\lambda(v_i)) \subseteq \delta_u(\lambda(u))$ and hence the claim holds. So we assume that r is of the form ($\bot$-`Norm`). Then it suffices to show that $\mathsf{overlap}(b_q, b_\ell)$ with $\ell = N_\bot(r)$.

- If $\lambda(u) \neq \Box$, there exists i such that χ_i is not empty. Let i be such a number. If $R \in \delta_{v_i}(\chi_i)$, then $\{R, \bot_\ell\} \subseteq \delta_{v_i}(\lambda(v_i))$ and hence $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_\ell)$ is entailed by Lemma A.18. Otherwise, there exists k that $\bot_k \in \delta_{v_i}(\chi_i)$. Then $\{\bot_k, \bot_\ell\} \subseteq \delta_{v_i}(\lambda(v_i))$. By induction hypothesis, we have $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_k)$. By Lemma A.16, we know $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_\ell, b_k)$. Because $\mathsf{overlap}$ is transitive and symmetric, $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_\ell)$.

- Otherwise, $q = \bot$. By Definition A.5, the fact $P^R_\bot$ and the rule $P^R_\bot \wedge \mathsf{body}(r) \to \mathsf{overlap}(b_q, b_\ell)$ are in $\mathsf{track}(\mathcal{K}, q, G)$. By Proposition A.12, we have $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_\ell)$. $\Box$

Now we are ready the prove the main Lemma A.7.

Proof of Lemma A.7. For each fact $\alpha \in \mathsf{support}(\rho)$, there is a leaf v such that $\lambda(v) = \alpha$. If $\delta_v(\alpha) = \{R\}$, then $\mathsf{track}(\mathcal{K}, q, G) \models (\alpha\tau)^R$ by Lemma A.13 and hence $\alpha \in \mathcal{K}^{q,G}$; otherwise, $\bot_\ell \in \delta_v(\alpha)$ and hence $\alpha \in \mathcal{K}^{\bot_\ell} \subseteq \mathcal{K}^{q,G}$ since $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_\ell)$ is implied by Lemma A.20. Therefore, all facts in $\mathsf{support}(\rho)$ are contained in $\mathcal{K}^{q,G}$.

We next prove that each rule that corresponds to the main premise of a hyperresolution step in ρ is contained in $\mathcal{K}^{q,G}$. Consider a hyperresolution step in ρ. Let r be the rule in $\mathcal{K} \cup \mathcal{R}_q$ such that $\mathsf{sk}(r)$ is the main premise in the hyperresolution step to obtain $\lambda(u)$ with MGU σ, i.e. $\lambda(u) = \gamma_1\sigma \vee \cdots \vee \gamma_m\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is the hyperresolvent of $\lambda(v_i) = \beta_i\sigma \vee \chi_i$ for $1 \leq i \leq n$ and $\mathsf{sk}(r) = \neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \gamma_1 \vee \cdots \vee \gamma_m$.

- If there exists j such that $\delta_u(\gamma_j\sigma) = \{R\}$, then $\mathsf{track}(\mathcal{K}, q, G) \models (\gamma_i\sigma\tau)^{R_A}$ for each $1 \leq i \leq n$ and $\mathsf{track}(\mathcal{K}, q, G) \models (\gamma_j\sigma\tau)^R$. By Definition A.5 and Lemma A.13, $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{Rel}(\mathsf{d}_r)$.

- If $m = 0$, let $\ell = N_\perp(r)$. For each $1 \leq i \leq n$, $\perp_\ell \in \lambda_{v_i}(\lambda(v_i))$. By Lemma A.20, we have $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_\ell)$ and hence $\mathcal{K}^{\perp_\ell} \subseteq \mathcal{K}^{q,G}$. The rule $\perp_\ell^{\perp_\ell} \wedge \mathsf{body}(r) \rightarrow \mathsf{Rel}_\ell(\mathsf{d}_r)$ in $\mathsf{track}_\perp(\mathcal{K})$. By Proposition A.12, we have $\mathsf{track}_\perp(\mathcal{K}) \models \mathsf{Rel}_\ell(\mathsf{d}_r)$ and hence $r \in \mathcal{K}^{\perp_\ell}$.

- Otherwise, a tuple $(k_1, \ldots, k_n)$ exists that $\perp_{k_i} \in \delta_u(\gamma_i\sigma)$. By Lemma A.13 and Proposition A.9, we have $\mathsf{track}_\perp(\mathcal{K}) \models (\gamma_i\sigma\tau)^{\perp_{k_i}}$. Since $\mathsf{track}_\perp(\mathcal{K}) \models P^{\perp_\ell}(\mathbf{x}) \rightarrow P^{\perp_A}(\mathbf{x})$ for each $1 \leq \ell \leq N$, we have $\mathsf{track}_\perp(\mathcal{K}) \models (\gamma_i\sigma\tau)^{\perp_A}$ for each i. Let $k = k_j$ for any j. Because the rule $\bigwedge_{i \neq k} \gamma_i^{\perp_A} \wedge \gamma_j^{\perp_k} \wedge \mathsf{body}(r) \rightarrow \mathsf{Rel}_k(\mathsf{d}_r)$ in $\mathsf{track}_\perp(\mathcal{K})$. By Proposition A.12, we have $\mathsf{track}_\perp(\mathcal{K}) \models \mathsf{Rel}_k(\mathsf{d}_r)$ and hence $r \in \mathcal{K}^{\perp_k}$. By Lemma A.20, $\mathsf{track}(\mathcal{K}, q, G) \models \mathsf{overlap}(b_q, b_k)$ and hence $r \in \mathcal{K}^{q,G}$.

Therefore, $r \in \mathcal{K}^{q,G}$ for each $r \in \mathsf{support}(\rho)$. □

Bibliography

Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

Andrea Acciarri, Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati. QuOnto: Querying ontologies. In *AAAI 2005, Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 1670–1671. AAAI Press / The MIT Press, 2005.

Mario Alviano, Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Magic sets for disjunctive datalog programs. *Artificial Intelligence*, 187–188:156–192, 2012a.

Mario Alviano, Wolfgang Faber, Nicola Leone, and Marco Manna. Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues. *Theory and Practice of Logic Programming*, 12(4-5):701–718, 2012b.

Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, and Dmitriy Zheleznyakov. Faceted search over ontology-enhanced RDF data. In *CIKM 2014, Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, Shanghai, China, November 3-7, 2014*, pages 939–948. ACM, 2014.

Ana Armas Romero, Mark Kaminski, Bernardo Cuenca Grau, and Ian Horrocks. Ontology module extraction via datalog reasoning. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1410–1416. AAAI Press, 2015.

Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The *DL-Lite* family and relations. *Journal of Artificial Intelligence Research*, 36:1–69, 2009.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge Univ. Press, 2003.

Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *IJCAI 2005, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 364–369, 2005.

Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 19–99. Elsevier and MIT Press, 2001.

Timea Bagosi, Diego Calvanese, Josef Hardi, Sarah Komla-Ebri, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, Mindaugas Slusnys, and Guohui Xiao. The Ontop framework for ontology based data access. In *CSWS 2014, Proceedings of the Semantic Web and Web Science - 8th Chinese Conference, Wuhan, China, August 8-12, 2014, Revised Selected Papers*, volume 480 of *Communications in Computer and Information Science*, pages 67–77. Springer, 2014.

François Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 24-26, 1986, Cambridge, Massachusetts, USA*, pages 1–15. ACM, 1986.

Pablo Barceló, Leonid Libkin, and Miguel Romero. Efficient approximations of conjunctive queries. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 249–260. ACM, 2012.

Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

Catriel Beeri, Shamim A. Naqvi, Raghu Ramakrishnan, Oded Shmueli, and Shalom Tsur. Sets and negation in a logic database language (LDL1). In Moshe Y. Vardi, editor, *Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 23-25, 1987, San Diego, California, USA*, pages 21–37. ACM, 1987.

Philippe Besnard. *An introduction to default logic*. Symbolic computation. Springer, 1989. ISBN 978-3-540-51566-1.

Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1):33–42, 2011.

Pierre Bourhis, Michael Morak, and Andreas Pieris. The impact of disjunction on query answering under guarded-based existential rules. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 796–802. AAAI Press, 2013.

Dan Brickley and R.V. Guha. RDF schema 1.1. W3C Recommendation, 25 February 2014. Available at http://www.w3.org/TR/rdf-schema/.

Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *ISWC 2002, Proceedings the Semantic Web - First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, volume 2342 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2002.

Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

Andrea Calì, Georg Gottlob, and Andreas Pieris. New expressive languages for ontological query answering. In *AAAI 2011, Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, San Francisco, California, USA, August 7-11, 2011*, volume 2, pages 1541–1546. AAAI Press, 2011.

Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics*, 14:57–83, 2012.

Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research*, 48:115–174, 2013.

Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Modeling and querying semi-structured data. *Networking and Information Systems*, 2(2):253–273, 1999.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi. Reasoning in expressive description logics. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1581–1634. Elsevier and MIT Press, 2001.

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. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The MASTRO system for ontology-based data access. *Semantic Web*, 2(1):43–53, 2011.

Yang Cao, Wenfei Fan, Tianyu Wo, and Wenyuan Yu. Bounded conjunctive queries. *PVLDB*, 7(12):1231–1242, 2014.

Alexandros Chortaras, Despoina Trivela, and Giorgos B. Stamou. Optimized query rewriting for OWL 2 QL. In *CADE 23, Proceedings of the 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011*, volume 6803 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2011.

Marco Console, José Mora, Riccardo Rosati, Valerio Santarelli, and Domenico Fabio Savo. Effective computation of maximal sound approximations of description logic ontologies. In *ISWC 2014, Proceedings of the Semantic Web - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II*, pages 164–179, 2014.

Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Journal of Web Semantics*, 6(4):309–322, 2008.

Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *Journal of Artificial Intelligence Research*, 47:741–808, 2013.

Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Edith Schonberg, Kavitha Srinivas, and Li Ma. Scalable semantic retrieval through summarization and refinement. In *AAAI 2007, Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 299–304. AAAI Press, 2007.

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

H.D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Undergraduate Texts in Mathematics. Springer New York, 1996.

Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. Simplifying logic programs under uniform and strong equivalence. In *LPNMR 2004, Proceedings of Logic Programming and Nonmonotonic Reasoning - 7th International Conference, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings*, pages 87–99, 2004.

Thomas Eiter, Georg Gottlob, Magdalena Ortiz, and Mantas Simkus. Query answering in the description logic Horn-$\mathcal{SHIQ}$. In *JELIA 2008, Proceedings of Logics in Artificial Intelligence, 11th European Conference, Dresden, Germany, September 28 - October 1, 2008*, volume 5293 of *Lecture Notes in Computer Science*, pages 166–179. Springer, 2008.

Thomas Eiter, Carsten Lutz, Magdalena Ortiz, and Mantas Simkus. Query answering in description logics with transitive roles. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 759–764, 2009.

Thomas Eiter, Magdalena Ortiz, and Mantas Simkus. Conjunctive query answering in the description logic $\mathcal{SH}$ using knots. *Journal of Computer and System Sciences*, 78(1):47–85, 2012.

Orri Erling and Ivan Mikhailov. Virtuoso: RDF support in a native RDBMS. In *Semantic Web Information Management - A Model-Based Perspective*, pages 501–519. Springer, 2009.

Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1, 1999.

Robert Fink, Jiewen Huang, and Dan Olteanu. Anytime approximation in probabilistic databases. *VLDB J.*, 22(6):823–848, 2013.

Wolfgang Gatterbauer and Dan Suciu. Approximate lifted inference with probabilistic databases. *PVLDB*, 8(5):629–640, 2015.

Birte Glimm and Chimezie Ogbuji, editors. *SPARQL 1.1 Entailment Regimes*. World Wide Web Consortium, January 2012. Published: W3C Working Draft Available at http://www.w3.org/TR/sparql11-entailment/.

Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. *Journal of Artificial Intelligence Research*, 31:157–204, 2008.

Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. HermiT: An OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.

Birte Glimm, Yevgeny Kazakov, Ilianna Kollia, and Giorgos Stamou. Lower and upper bounds for sparql queries over owl ontologies. In *AAAI 2015, Proceedings of the 29th AAAI Conference on Artificial Intelligence*. AAAI Press, 2015.

Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *WWW 2003, Proceedings of the Twelfth International World Wide Web Conference, Budapest, Hungary, May 20-24, 2003*, pages 48–57. ACM, 2003.

Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.

Volker Haarslev and Ralf Möller. On the scalability of description logic instance retrieval. *Journal of Automated Reasoning*, 41(2):99–142, 2008.

Volker Haarslev, Ralf Möller, and Michael Wessel. Querying the semantic web with racer + nRQL. In *Proc. of the KI-2004 Intl. Workshop on Applications of Description Logics (ADL'04)*, 2004.

Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The RacerPro knowledge representation and reasoning system. *Semantic Web*, 3(3):267–277, 2012.

Peter Hansen, Carsten Lutz, Inanç Seylan, and Frank Wolter. Query rewriting under $\mathcal{EL}$ TBoxes: Efficient algorithms. In Meghyn Bienvenu, Magdalena Ortiz, Riccardo Rosati, and Mantas Simkus, editors, *DL 2014, Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014.*, volume 1193 of *CEUR Workshop Proceedings*, pages 197–208. CEUR-WS.org, 2014.

Ian Horrocks and Sergio Tessaris. A conjunctive query language for description logic aboxes. In *AAAI/IAAI 2000, Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 399–404. AAAI Press / The MIT Press, 2000.

Ian Horrocks and Stephan Tobies. Reasoning with axioms: Theory and practice. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000.*, pages 285–296. Morgan Kaufmann, 2000.

Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.

Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: the making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *KR 2006, Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 57–67, 2006.

Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in description logics by a reduction to disjunctive datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.

Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. LogMap: Logic-based and scalable ontology matching. In *ISWC 2011, The Semantic Web - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2011.

Yevgeny Kazakov. $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In *KR 2008, Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, Sydney, Australia, September 16-19, 2008*, pages 274–284. AAAI Press, 2008.

Yevgeny Kazakov. Consequence-driven reasoning for horn $\mathcal{SHIQ}$ ontologies. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 2040–2045, 2009.

Yevgeny Kazakov. An extension of complex role inclusion axioms in the description logic SROIQ. In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*, pages 472–486. Springer, 2010.

David B. Kemp, Divesh Srivastava, and Peter J. Stuckey. Bottom-up evaluation and query optimization of well-founded models. *Theoretical Computer Science*, 146 (1–2):145–184, 1995.

Ilianna Kollia and Birte Glimm. Optimizing SPARQL query answering over OWL ontologies. *Journal of Artificial Intelligence Research*, 48:253–303, 2013.

Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to ontology-based data access. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2656–2661. IJCAI/AAAI, 2011.

Lee Lacy, Gabriel Aviles, Karen Fraser, William Gerber, Alice M. Mulvehill, and Robert Gaskill. Experiences using OWL in military applications. In Bernardo Cuence Grau, Ian Horrocks, Bijan Parsia, and Peter F. Patel-Schneider, editors, *Proceedings of the OWLED 05 Workshop on OWL: Experiences and Directions, Galway, Ireland, November 11-12, 2005*, volume 188 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, July 2006. ISSN 1529-3785.

Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *KDD 2006, Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 631–636, 2006.

Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994*, pages 23–37. MIT Press, 1994.

John W. Lloyd. *Foundations of Logic Programming, 1st Edition*. Springer, 1984. ISBN 3-540-13299-6.

Carsten Lutz. The complexity of conjunctive query answering in expressive description logics. In *IJCAR 2008, Proceedings of the 4th International Joint Conference Automated Reasoning, Sydney, Australia, August 12-15, 2008*, volume 5195 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2008.

Carsten Lutz, David Toman, and Frank Wolter. Conjunctive query answering in the description logic EL using a relational database system. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 2070–2075, 2009.

Carsten Lutz, Inanç Seylan, David Toman, and Frank Wolter. The combined approach to OBDA: Taming role hierarchies using filters. In *ISWC 2013, Proceedings of the Semantic Web - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, volume 8218 of *Lecture Notes in Computer Science*, pages 314–330. Springer, 2013.

Li Ma, Yang Yang, Zhaoming Qiu, Guo Tong Xie, Yue Pan, and Shengping Liu. Towards a complete OWL ontology benchmark. In *ESWC 2006, The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, Budva, Montenegro, June 11-14, 2006, Proceedings*, volume 4011 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2006.

Frank Manola and Eric Miller. RDF primer. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/rdf-primer/.

Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *PODS 2009, Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, June 19 - July 1, 2009, Providence, Rhode Island, USA*, pages 13–22, 2009.

Brian McBride. Jena: Implementing the RDF model and syntax specification. In *SemWeb 2001, Proceedings of the Second International Workshop on the Semantic Web*, 2001.

Ralf Möller, Christian Neuenstadt, Özgür L. Özcep, and Sebastian Wandelt. Advances in accessing big data with expressive ontologies. In *KI 2013, Proceedings of Advances in Artificial Intelligence - 36th Annual German Conference on AI, Koblenz, Germany, September 16-20, 2013*, volume 8077 of *Lecture Notes in Computer Science*, pages 118–129. Springer, 2013.

Boris Motik and Ulrike Sattler. A comparison of reasoning techniques for querying large description logic ABoxes. In Miki Hermann and Andrei Voronkov, editors, *LPAR 2006, Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, Phnom Penh, Cambodia, November 13-17, 2006, Proceedings*, volume 4246 of *Lecture Notes in Computer Science*, pages 227–241. Springer, 2006.

Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles. W3C Recommendation, 27 October 2009a. Available at http://www.w3.org/TR/owl2-profiles/.

Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language Structural Specification and Functional-style Syntax. W3C Recommendation, 27 October 2009b. Available at http://www.w3.org/TR/owl2-syntax/.

Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009c.

Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In *AAAI 2014, Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 129–137. AAAI Press, 2014.

Thomas Neumann and Gerhard Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB Journal*, 19(1):91–113, 2010.

Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 335–367. Elsevier and MIT Press, 2001.

Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data complexity of query answering in expressive description logics via tableaux. *Journal of Automated Reasoning*, 41(1):61–98, 2008.

Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Query answering in the horn fragments of the description logics $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 1039–1044, 2011.

David Osumi-Sutherland, Simon Reeve, Christopher J. Mungall, Fabian Neuhaus, Alan Ruttenberg, Gregory S. X. E. Jefferis, and J. Douglas Armstrong. A strategy for building neuroanatomy ontologies. *Bioinformatics*, 28(9):1262–1269, 2012.

Jeff Z. Pan and Edward Thomas. Approximating OWL-DL ontologies. In *AAAI 2007, Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 1434–1439, 2007.

Jeff Z. Pan, Edward Thomas, and Yuting Zhao. Completeness guaranteed approximations for OWL-DL query answering. In *DL 2009, Proceedings of the 22nd International Workshop on Description Logics Oxford, UK, July 27-30, 2009*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, 8(2): 186–209, 2010.

Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *Journal of Data Semantics*, 4900:133–173, 2008.

Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C Recommendation, 15 January 2008. Available at http://www.w3.org/TR/rdf-sparql-query/.

John Alan Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1(3):227, 1965.

John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

Riccardo Rosati. On conjunctive query answering in EL. In *DL 2007, Proceedings of the 2007 International Workshop on Description Logics, Brixen-Bressanone, near Bozen-Bolzano, Italy, 8-10 June, 2007*, volume 250 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

Riccardo Rosati. Prexto: Query rewriting under extensional constraints in DL-Lite. In *ESWC 2012, Proceedings of the Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, Heraklion, Crete, Greece, May 27-31, 2012*, volume 7295 of *Lecture Notes in Computer Science*, pages 360–374. Springer, 2012.

Sebastian Rudolph and Birte Glimm. Nominals, inverses, counting, and conjunctive queries or: Why infinity is your friend! *Journal of Artificial Intelligence Research*, 39:429–481, 2010.

Konstantinos F. Sagonas and Terrance Swift. An abstract machine for tabled execution of fixed-order stratified logic programs. *ACM Transactions on Programming Languages and Systems*, 20(3):586–634, 1998.

Andrea Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. In *ISMIS 1993, Proceedings of Methodologies for Intelligent Systems, 7th International Symposium, Trondheim, Norway, June 15-18, 1993*, volume 689 of *Lecture Notes in Computer Science*, pages 508–517. Springer, 1993.

Stefan Schulz, Ronald Cornet, and Kent A. Spackman. Consolidating SNOMED ct's ontological commitment. *Applied Ontology*, 6(1):1–11, 2011.

Bart Selman and Henry A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.

Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2): 51–53, 2007.

Kent A. Spackman and Keith E. Campbell. Compositional concept representation using SNOMED: towards further convergence of clinical terminologies. In *AMIA 1998, American Medical Informatics Association Annual Symposium, Lake Buena Vista, FL, USA, November 7-11, 1998*. AMIA, 1998.

Giorgio Stefanoni and Boris Motik. Answering conjunctive queries over EL knowledge bases with transitive and reflexive roles. In *AAAI 2015, Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 129–137, Austin, TX, USA, January 25–30 2015. AAAI Press. To appear.

Giorgio Stefanoni, Boris Motik, and Ian Horrocks. Introducing nominals to the combined query answering approaches for $\mathcal{EL}$. In *AAAI 2013, Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 1177–1183, 2013.

Giorgio Stefanoni, Boris Motik, Markus Krötzsch, and Sebastian Rudolph. The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. *Journal of Artificial Intelligence Research*, 51:645–705, 2014.

Markus Stocker and Michael Smith. Owlgres: A scalable OWL reasoner. In *ISWC 2008, Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference, Karlsruhe, Germany, October 26-27, 2008*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

Giorgos Stoilos. Ontology-based data access using rewriting, OWL 2 RL systems and repairing. In Valentina Presutti, Claudia d'Amato, Fabien Gandon, Mathieu d'Aquin, Steffen Staab, and Anna Tordai, editors, *ESWC 2014, Proceedings of the Semantic Web: Trends and Challenges - 11th International Conference, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, volume 8465 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2014a.

Giorgos Stoilos. Hydrowl: A hybrid query answering system for OWL 2 DL ontologies. In *RR 2014, Proceedings of Web Reasoning and Rule Systems - 8th International Conference, Athens, Greece, September 15-17, 2014*, pages 230–238, 2014b.

Giorgos Stoilos and Giorgos B. Stamou. Hybrid query answering over OWL ontologies. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014, Proceedings of the 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 855–860. IOS Press, 2014.

Hisao Tamaki and Taisuke Sato. OLD resolution with tabulation. In *Third International Conference on Logic Programming, Imperial College of Science and Technology, London, United Kingdom, July 14-18, 1986, Proceedings*, volume 225 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 1986.

Edward Thomas, Jeff Z. Pan, and Yuan Ren. Trowl: Tractable OWL 2 reasoning infrastructure. In *ESWC 2010, Proceedings of the Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, Heraklion, Crete, Greece, May 30 - June 3, 2010, Part II*, pages 431–435, 2010.

Stephan Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *Journal of Artificial Intelligence Research*, 12: 199–217, 2000.

Tuvshintur Tserendorj, Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. Approximate OWL-reasoning with screech. In *RR 2008, Proceedings of Web Reasoning and Rule Systems, Second International Conference, Karlsruhe, Germany, October 31-November 1, 2008*, volume 5341 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2008.

Alvaro del Val. First order LUB approximations: characterization and algorithms. *Artificial Intelligence*, 162(1-2):7–48, 2005.

W3C SPARQL Working Group. SPARQL 1.1 Overview. W3C Recommendation, 21 March 2013. Available at http://www.w3.org/TR/sparql11-overview/.

Sebastian Wandelt, Ralf Möller, and Michael Wessel. Towards scalable instance retrieval over ontologies. *International Journal of Software and Informatics*, 4(3): 201–218, 2010.

Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: sextuple indexing for semantic web data management. *PVLDB*, 1(1):1008–1019, 2008.

Zhe Wu, George Eadon, Souripriya Das, Eugene Inseok Chong, Vladimir Kolovski, Melliyal Annamalai, and Jagannathan Srinivasan. Implementing an inference engine for RDFS/OWL constructs and user-defined rules in oracle. In *ICDE 2008, Proceedings of the 24th International Conference on Data Engineering, April 7-12, 2008, Cancún, México*, pages 1239–1248. IEEE, 2008.

Yujiao Zhou, Bernardo Cuence Grau, Ian Horrocks, Zhe Wu, and Jay Banerjee. Making the most of your triple store: query answering in OWL 2 using an RL reasoner. In *WWW 2013, Proceedings of 22nd International World Wide Web Conference, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 1569–1580. International World Wide Web Conferences Steering Committee / ACM, 2013a.

Yujiao Zhou, Yavor Nenov, Bernardo Cuence Grau, and Ian Horrocks. Complete query answering over horn ontologies using a triple store. In *ISWC 2013, The Semantic Web - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, volume 8218 of *Lecture Notes in Computer Science*, pages 720–736. Springer, 2013b.

Yujiao Zhou, Yavor Nenov, Bernardo Cuence Grau, and Ian Horrocks. Pay-as-you-go OWL query answering using a triple store. In Carla E. Brodley and Peter Stone, editors, *AAAI 2014, Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 1142–1148. AAAI Press, 2014.

Index