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

Yujiao Zhou YUJIAO.ZHOU@CS.OX.AC.UK
Bernardo Cuenca Grau BERNARDO.CUENCA.GRAU@CS.OX.AC.UK
Yavor Nenov YAVOR.NENOV@CS.OX.AC.UK
Mark Kaminski MARK.KAMINSKI@CS.OX.AC.UK
Ian Horrocks IAN.HORROCKS@CS.OX.AC.UK
Department of Computer Science, University of Oxford
Parks Road, Oxford OX1 3QD, United Kingdom

Abstract

Answering conjunctive queries over ontology-enriched datasets is a core reasoning task for many applications of semantic technologies. Query answering is, however, computationally very expensive, which has led to the development of query answering procedures that sacrifice either expressive power of the ontology language, or the completeness of query answers in order to improve scalability. In this paper, we describe 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 our 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. Furthermore, although our main goal is to efficiently answer queries over OWL 2 ontologies and data, our technical results are very general and our approach is applicable to first-order knowledge representation languages that can be captured by rules allowing for existential quantification and disjunction in the head; our 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'. We have implemented our techniques in the PAGOdA system, which combines the datalog reasoner RDFox and the OWL 2 reasoner HermiT. Our 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.

1. Introduction

Ontologies are increasingly used as rich conceptual schemas in a wide range of application domains (Staab & Studer, 2004). One of the most widely used ontology languages is OWL, a description logic based language that was standardised by the World Wide Web Consortium (W3C) in 2004 and revised (as OWL 2) in 2009 (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003; Horrocks, Patel-Schneider, & van Harmelen, 2003; Cuenca Grau, Horrocks, Motik, Parsia, Patel-Schneider, & Sattler, 2008). An OWL ontology consists of a set of axioms, which correspond to first-order sentences containing only unary and binary predicates (called classes and properties in OWL), with the structure of axioms/sentences being restricted to ensure the decidability of basic reasoning problems.

In some applications, the main focus is on the conceptual model itself, with class subsumption being a key reasoning problem. In an increasing number of applications, however, the main focus is on using the conceptual model to access data, often in the form of an RDF graph (Manola & Miller, 2004). In such data-centric applications a key reasoning problem is to answer conjunctive queries (CQs)—sentences constructed from function-free atoms using conjunction and existential quantification only (Abiteboul, Hull, & Vianu, 1995)—which constitute the core component of standard query languages such as SQL and SPARQL (W3C SPARQL Working Group, 2013).

Conjunctive query answering over ontology-enriched datasets is, however, of high worst-case complexity (Glimm, Lutz, Horrocks, & Sattler, 2008; Eiter, Ortiz, & Simkus, 2012), even when measured only with respect to the size of the data (so called *data complexity*). Although heavily optimised, existing systems for query answering with respect to (RDF) data and an unrestricted OWL 2 ontology can process only small to medium size datasets (Sirin, Parsia, Cuenca Grau, Kalyanpur, & Katz, 2007; Möller, Neuenstadt, Özcep, & Wandelt, 2013; Wandelt, Möller, & Wessel, 2010; Kollia & Glimm, 2013). This has led to the development of query answering procedures that sacrifice expressive power of the ontology language or the completeness of query answers in order to improve scalability.

In the former case (sacrificing expressive power), query answering procedures have been developed for various fragments of OWL 2 for which conjunctive query answering is tractable with respect to data complexity, and three such fragments were standardised as so-called *profiles* in OWL 2 (Motik, Cuenca Grau, Horrocks, Wu, Fokoue, & Lutz, 2009). The OWL 2 QL and OWL 2 EL profiles are based on the DL-Lite (Calvanese, De Giacomo, Lembo, Lenzerini, & Rosati, 2007) and $\mathcal{EL}$ (Baader, Brandt, & Lutz, 2005) families of description logics; the OWL 2 RL profile corresponds to a fragment of the rule-based language datalog (Grosof, Horrocks, Volz, & Decker, 2003; Dantsin, Eiter, Gottlob, & Voronkov, 2001). Conjunctive Query answering systems for such profiles have been shown to be highly scalable in practice (Bishop, Kiryakov, Ognyanoff, Peikov, Tashev, & Velkov, 2011; Wu, Eadon, Das, Chong, Kolovski, Annamalai, & Srinivasan, 2008; Motik, Nenov, Piro, Horrocks, & Olteanu, 2014; Erling & Mikhailov, 2009; Rodriguez-Muro & Calvanese, 2012; Lutz, Seylan, Toman, & Wolter, 2013; Stefanoni, Motik, & Horrocks, 2013). The more favourable computational properties of these fragments make them a natural choice for data-intensive applications, but they also come at the expense of a loss in expressive power, and many ontologies used in applications are not captured by any of the profiles.

In the latter case (sacrificing completeness), query answering procedures have been developed that exploit scalable reasoning techniques, but at the expense of computing only approximate query answers (Thomas, Pan, & Ren, 2010; Tserendorj, Rudolph, Krötzsch, & Hitzler, 2008; Wandelt et al., 2010; Bishop et al., 2011). In most cases, the computed answers are sound (only correct answer tuples are identified) but incomplete (some correct answer tuples may not be identified). One way to realise such a procedure is to weaken the ontology until it falls within one of the OWL 2 profiles, and then to use a scalable procedure for the relevant fragment. The required weakening can be trivially achieved simply by discarding (parts of) out-of-profile axioms, but more sophisticated techniques may try to reduce or even minimise information loss (Console, Mora, Rosati, Santarelli, & Savo, 2014). Such an approach is clearly sound (if an answer tuple is entailed by the weakened ontology, then it is entailed by the original ontology), but incomplete in general, and for ontologies outside the relevant profile, the answer returned by such systems can therefore be understood as providing a *lower-bound* on the correct answer; however, such procedures cannot in general provide any complementary upper bound or even any indication as to how complete the computed answer is (Cuenca Grau, Motik, Stoilos, & Horrocks, 2012).

In this paper, we describe a novel hybrid approach to query answering that combines a scalable datalog (or OWL 2 RL) reasoner with a fully-fledged OWL 2 reasoner to provide scalable performance while still guaranteeing sound and complete answers in all cases. Our procedure uses the datalog reasoner to efficiently compute both lower bound (sound but possibly incomplete) and upper bound (complete but possibly unsound) answers to the input query. If lower and upper bound answers coincide, they obviously provide a sound and complete answer. Otherwise, *relevant subsets* of the ontology and data are computed that are guaranteed to be sufficient to test the correctness of tuples in the 'gap' between the lower and upper bounds. These subsets are computed using only the datalog reasoner, and they are typically much smaller than the input ontology and data. Finally, the fully-fledged reasoner is used to check gap tuples w.r.t. the relevant subset. As this can still be computationally expensive, the load on the fully-fledged reasoner is further reduced by exploiting summarisation techniques inspired by the SHER system to quickly identify spurious gap tuples (Dolby, Fokoue, Kalyanpur, Kershenbaum, Schonberg, Srinivas, & Ma, 2007; Dolby, Fokoue, Kalyanpur, Schonberg, & Srinivas, 2009), and by analysing dependencies between remaining gap tuples to reduce the number of checks that need to be performed.

The key feature of our approach is its 'pay-as-you-go' behaviour: the bulk of the computational workload is delegated to the datalog reasoner, and the extent to which the fully-fledged reasoner is needed does not depend solely on the ontology, but on interactions between the ontology, the dataset and the query. Thus, even when using a very expressive ontology, queries can often be fully answered using only the datalog reasoner, and even when the fully-fledged reasoner is required, relevant subset extraction, summarisation and dependency analysis greatly reduce the number and size of reasoning problems. Moreover, our approach has the additional advantage that lower bound answer tuples can be quickly returned, even in cases where completion of the answer requires more time consuming computations. Finally, although our main goal is to efficiently answer queries over OWL 2 ontologies and datasets, our technical results are very general and our approach is not restricted to ontology languages based on description logics. More precisely, given a KR language $\mathcal{L}$ that can be captured by first-order rules allowing for existential quantification and disjunction in the head, and over which we want to answer conjunctive queries, our only assumption is the availability of a fully-fledged reasoner for $\mathcal{L}$ and a datalog reasoner, both of which are used as a 'black box'.

We have implemented our techniques in the PAGOdA system[1] using RDFox as a datalog reasoner (Motik et al., 2014) and HermiT as a fully-fledged OWL 2 reasoner (Glimm, Horrocks, Motik, Stoilos, & Wang, 2014),[2] and conducted an extensive evaluation using a wide range of realistic and benchmark datasets and queries. This evaluation suggests that our techniques are very effective at providing scalable pay-as-you-go query answering: in our tests of more than 4,000 queries over 8 ontologies, none of which is contained within any of the OWL profiles, more than 99% of queries were fully answered without resorting to the fully-fledged reasoner. Moreover, even when the fully-fledged reasoner was used, relevant

1. `http://www.cs.ox.ac.uk/isg/tools/PAGOdA/`
2. Although our techniques are proved correct for general conjunctive queries, in practice we are limited by the current query capabilities of OWL 2 reasoners.

subset extraction, summarisation and dependency analysis greatly reduced the number and size of reasoning problems: in our tests, the size of the dataset was typically reduced by an order magnitude, and often by several orders of magnitude, and it seldom required more than a single test to resolve the status of all gap tuples. Taken together, our experiments demonstrate that PAGOdA can provide an efficient conjunctive query answering service in real-world scenarios requiring both expressive ontologies and datasets containing hundreds of millions of facts, something that is far beyond the capabilities of pre-existing state-of-the-art ontology reasoners.

The remainder of the paper is organised as follows. In Section 2 we introduce key concepts and definitions. In Section 3 we present a high-level overview of our approach. In Section 4 we describe how lower bound answers are computed and prove that they are sound, and in Section 5 we describe how upper bound answers are computed and prove that they are complete. In Section 6 we present our technique for reducing the size of the ontology and dataset to be processed by the fully-fledged reasoner and prove that it preserves completeness. In Section 7 we present our summarisation and dependency analysis optimisations and prove that they too preserve completeness. In Section 8 we describe the implementation of our techniques in the PAGOdA system and discuss some additional optimisations. Finally, after positioning our work within the state-of-the-art in Section 9, we present our extensive evaluation in Section 10, and draw our conclusions in Section 11.

2. Preliminaries

In this section we briefly introduce rule-based first-order languages and description logics (DLs)—a family of knowledge representation formalisms underpinning the OWL and OWL 2 ontology languages (Baader et al., 2003).

We use standard notions from first-order logic such as *constant*, *predicate*, *function*, *term*, *substitution*, *atom*, *formula*, and *sentence*. We also adopt standard definitions of *(Herbrand) interpretation* and *model*, as well as of *(un)satisfiability* and *entailment* (written $\models$) of sets of first-order sentences. We denote with $\bot$ the nullary predicate that is false in all interpretations. Formulas may also contain the special equality predicate $\approx$. We assume that each first-order knowledge base $\mathcal{F}$ over a function-free signature that uses $\approx$ axiomatises its semantics in the usual way; that is, $\mathcal{F}$ must contain the following first-order sentences, where (EQ1) and (EQ4) are instantiated for each n-ary predicate P in $\mathcal{F}$ and each $1 \leq i \leq n$:

$$\forall x_1, \ldots, x_n (P(x_1, \ldots, x_i, \ldots, x_n) \rightarrow x_i \approx x_i) \tag{EQ1}$$

$$\forall x, y (x \approx y \rightarrow y \approx x) \tag{EQ2}$$

$$\forall x, y, z (x \approx y \wedge y \approx z \rightarrow x \approx z) \tag{EQ3}$$

$$\forall x_1, \ldots, x_n, y (P(x_1, \ldots, x_i, \ldots, x_n) \wedge x_i \approx y \rightarrow P(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)) \tag{EQ4}$$

Finally, we will also exploit the following notion of *homomorphism* applicable to sets of atoms, formulas and substitutions. Given sets of ground atoms S and T, we define a homomorphism from S to T as a mapping τ from ground terms to ground terms s.t. $\tau(c) = c$ for any constant c in S and T, and $P(t_1\tau, \ldots, t_n\tau) \in T$ for each atom $P(t_1, \ldots, t_n) \in S$. The application of a homomorphism can be naturally extended to ground atoms, ground formulas and ground substitutions, e.g. for an atom $\alpha = P(t_1, \ldots, t_n)$, $\alpha\tau = P(t_1\tau, \ldots, t_n\tau)$ and for a ground substitution σ, $\sigma\tau$ is the substitution $\{x \mapsto x\sigma\tau \mid x \in \mathsf{dom}(\sigma)\}$.

2.1 Rule-based Knowledge Representation

Rule languages are well-known knowledge representation formalisms which are strongly connected with ontology languages (Dantsin et al., 2001; Calì, Gottlob, Lukasiewicz, Marnette, & Pieris, 2010; ?).

We define a *fact* as a function-free ground atom and a *dataset* as a finite set of facts. A *rule* r is a function-free first-order sentence of the form

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

where each $\beta_i(\vec{x}, \vec{y})$ is an atom different from $\bot$ with free variables in $\vec{x} \cup \vec{y}$, and either

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

The conjunction of atoms $\beta_1(\vec{x}, \vec{y}) \wedge \cdots \wedge \beta_n(\vec{x}, \vec{y})$ is the *body* of r, denoted by $\mathsf{body}(r)$. The formula $\bigvee_{i=1}^{m} \exists \vec{z}_i \varphi_i(\vec{x}, \vec{z}_i)$ is the *head* of r, denoted by $\mathsf{head}(r)$. We assume that rules are *safe*; that is, every variable in $\vec{x}$ is mentioned in $\mathsf{body}(r)$. For brevity, universal quantifiers are omitted in rules.

Rules of this form are very general and are able to capture most first-order rule languages for knowledge representation, including datalog (Abiteboul et al., 1995), existential rules and datalog$^{\pm}$ (Calì et al., 2010), as well as datalog$^{\pm,\vee}$ (Alviano, Faber, Leone, & Manna, 2012; Bourhis, Morak, & Pieris, 2013).

We say that a rule r is

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

A *knowledge base* $\mathcal{K} = \Sigma_{\mathcal{K}} \cup \mathcal{D}_{\mathcal{K}}$ consists of a finite set of rules $\Sigma_{\mathcal{K}}$ and a dataset $\mathcal{D}_{\mathcal{K}}$ where each predicate in $\mathcal{D}_{\mathcal{K}}$ is assumed to occur in $\Sigma_{\mathcal{K}}$.

In order to simplify the presentation of our technical results, we sometimes restrict ourselves to knowledge bases in a particular normal form, which we specify next. We say that a rule r is *normalised* if it is of one of the following forms, where $m \geq 1$ and each $\gamma_i(\vec{x}, \vec{z}_i)$ is a single atom different from $\bot$:

$$\beta_1(\vec{x}, \vec{y}) \wedge \cdots \wedge \beta_n(\vec{x}, \vec{y}) \rightarrow \bot \tag{2}$$

$$\beta_1(\vec{x}, \vec{y}) \wedge \cdots \wedge \beta_n(\vec{x}, \vec{y}) \rightarrow \exists \vec{z}_1 \gamma_1(\vec{x}, \vec{z}_1) \tag{3}$$

$$\beta_1(\vec{x}, \vec{y}) \wedge \cdots \wedge \beta_n(\vec{x}, \vec{y}) \rightarrow \gamma_1(\vec{x}) \vee \cdots \vee \gamma_m(\vec{x}) \tag{4}$$

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 (1) can be transformed in polynomial time into a set of normalised rules $\mathsf{norm}(\Sigma)$ that is a conservative extension of Σ as given next. For each rule $r \in \Sigma$ and each $1 \leq i \leq m$, let $\vec{x}_i$ be the tuple of free variables in the subformulas $\exists \vec{z}_i \varphi_i(\vec{x}, \vec{z}_i)$ of $\mathsf{head}(r)$, then $\vec{x}_i \subseteq \vec{x}$. Furthermore, let E_{φ_i} be fresh predicates of arity $|\vec{x}_i|$ and let C_{φ_i} be fresh predicates of arity $|\vec{x}_i| + |\vec{z}_i|$ uniquely associated to r and i. Then, $\mathsf{norm}(\Sigma)$ consists of the following rules:[3]

$$\beta_1(\vec{x}, \vec{y}) \wedge \cdots \wedge \beta_n(\vec{x}, \vec{y}) \to \bigvee_{i=1}^{m} E_{\varphi_i}(\vec{x}_i), \tag{5}$$

$$E_{\varphi_i}(\vec{x}_i) \to \exists \vec{z}_i \, C_{\varphi_i}(\vec{x}_i, \vec{z}_i) \text{ for each } 1 \leq i \leq m, \tag{6}$$

$$C_{\varphi_i}(\vec{x}_i, \vec{z}_i) \to \gamma \text{ for each } 1 \leq i \leq m \text{ and each atom } \gamma \text{ in } \varphi_i(\vec{x}, \vec{z}_i), \tag{7}$$

$$\varphi_i(\vec{x}, \vec{z}_i) \to E_{\varphi_i}(\vec{x}_i) \text{ for each } 1 \leq i \leq m, \tag{8}$$

$$\varphi_i(\vec{x}, \vec{z}_i) \to C_{\varphi_i}(\vec{x}_i, \vec{z}_i) \text{ for each } 1 \leq i \leq m. \tag{9}$$

We frequently use *Skolemisation* to interpret rules in Herbrand interpretations. For each rule r of the form (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 $\vec{x}$. Furthermore, let θ_{sk} be the substitution such that $\theta_{\mathsf{sk}}(z_{ij}) = f^r_{ij}(\vec{x})$ for each $z_{ij} \in \vec{z}_i$. The Skolemisation $\mathsf{sk}(r)$ of r is the following first-order sentence, which by slight abuse of notation we refer to as a *Skolemised rule*:

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

The Skolemisation $\mathsf{sk}(\Sigma)$ of a set of rules Σ is obtained by Skolemising each individual rule in Σ. We extend the definitions of *head* and *body* of rules to Skolemised rules naturally. It is well-known that Skolemisation is an entailment-preserving transformation.

2.2 Description Logics and Ontology Languages

We next present a brief overview of the DLs underpinning the W3C standard ontology language OWL 2 (Horrocks, Kutz, & Sattler, 2006; Cuenca Grau et al., 2008). Typically, the predicates in DL signatures are restricted to be unary or binary; the former are called *atomic concepts*, whereas the latter are typically referred to as *atomic roles*. DLs typically provide two special concepts $\bot$ (the bottom concept) and $\top$ (the top concept), which are mapped by every interpretation to the empty set and the interpretation domain, respectively.

Every OWL 2 DL ontology can be normalised as a set of axioms of the form given on the left-hand-side of Table 1 (e.g., see (Motik, Shearer, & Horrocks, 2009)).[4] Thus, w.l.o.g. we define an *OWL 2 DL ontology* as a finite set of axioms of the form (O1)–(O13) in Table 1. Every OWL 2 DL ontology must satisfy certain additional requirements in order to ensure decidability of reasoning; these restrictions, however, are immaterial to our technical results and we refer the interested readers to (Horrocks et al., 2006) for details.

Each normalised axiom corresponds to a single rule, as given on the right-hand-side of Table 1. Concept $\bot$ is translated as the special nullary predicate $\bot$, whereas $\top$ is translated

3. Although rules (5)–(7) are sufficient to express Σ in normal form, we also introduce rules (8)–(9) in order to facilitate the computation of upper bound query answers (see Sections 5.2 and 5.3).
4. For convenience, 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	
$\bigsqcap_{i=1}^{n} A_i$	$\sqsubseteq \bot$	$\bigwedge_{i=1}^{n} A_i(x) \rightarrow \bot$	(O1)
$\bigsqcap_{i=1}^{n} A_i$	$\sqsubseteq \bigsqcup_{j=1}^{m} B_j$	$\bigwedge_{i=1}^{n} A_i(x) \rightarrow \bigvee_{j=1}^{m} B_j(x)$	(O2)
$\exists R.A$	$\sqsubseteq B$	$R(x,y) \wedge A(y) \rightarrow B(x)$	(O3)
A	$\sqsubseteq \mathsf{Self}(R)$	$A(x) \rightarrow R(x,x)$	(O4)
$\mathsf{Self}(R)$	$\sqsubseteq A$	$R(x,x) \rightarrow A(x)$	(O5)
R	$\sqsubseteq S$	$R(x,y) \rightarrow S(x,y)$	(O6)
R	$\sqsubseteq S^-$	$R(x,y) \rightarrow S(y,x)$	(O7)
$R \circ S$	$\sqsubseteq T$	$R(x,z) \wedge S(z,y) \rightarrow T(x,y)$	(O8)
$R \sqcap S$	$\sqsubseteq \bot$	$R(x,y) \wedge S(x,y) \rightarrow \bot$	(O9)
A	$\sqsubseteq \exists R.B$	$A(x) \rightarrow \exists y(R(x,y) \wedge B(y))$	(O10)
A	$\sqsubseteq \leq m\, R.B$	$A(x) \wedge \bigwedge_{i=1}^{m+1}[R(x,y_i) \wedge B(y_i)] \rightarrow \bigvee_{1 \leq i < j \leq m+1} y_i \approx y_j$	(O11)
A	$\sqsubseteq \{a\}$	$A(x) \rightarrow x \approx a$	(O12)
$\top$	$\sqsubseteq \forall R.A$	$R(x,y) \rightarrow A(y)$	(O13)

Table 1: Normalised DL axioms and their translation into rules where $n, m > 0$, A and B are atomic concepts or $\top$, and R, S, T are atomic roles.

as an ordinary unary predicate, the meaning of which is axiomatised. Let π be the function that maps an OWL 2 axiom α to its corresponding rule as in Table 1, and let $\mathcal{O}$ be an ontology. Then, $\pi(\mathcal{O})$ is the smallest knowledge base containing:

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

Note that since $\pi(\mathcal{O})$ is a knowledge base, it must contain the axioms of equality for its signature whenever $\approx$ is required to translate an axiom in $\mathcal{O}$.

We say that an ontology is $\mathcal{ELHO}^r_\bot$ if it does not contain axioms (O4), (O5), (O7), (O8), (O9) or (O11), and every axiom of type (O2) satisfies $m = 1$. Rules corresponding to $\mathcal{ELHO}^r_\bot$ axioms can be syntactically characterised as follows. A rule is $\mathcal{ELHO}^r_\bot$ if it is of one of the following forms, where $\varphi(x)$ is either $\bot$, or of the form $A(x)$, $x \approx c$, or $\exists y R(x,y)$:

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

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

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

2.3 Conjunctive Queries

A *conjunctive query (CQ)* is a formula $q(\vec{x})$ of the form $\exists \vec{y}\, \varphi(\vec{x}, \vec{y})$, where $\varphi(\vec{x}, \vec{y})$ is a conjunction of function-free atoms. A query is *Boolean* if $|\vec{x}| = 0$, and it is *atomic* if $\varphi(\vec{x}, \vec{y})$

consists of a single atom and $|\vec{y}| = 0$. For simplicity, we sometimes omit the free variables and write q instead of $q(\vec{x})$.

Let $\mathcal{K}$ be a knowledge base. A tuple $\vec{a}$ of constants is a *possible answer* to $q(\vec{x})$ w.r.t. $\mathcal{K}$ if it is of the same arity as $\vec{x}$ and each constant in $\vec{a}$ occurs in $\mathcal{K}$. Furthermore, we say that a possible answer $\vec{a}$ is a *certain answer* if $\mathcal{K} \models q(\vec{a})$; the set of such certain answers is denoted by $\mathsf{cert}(q, \mathcal{K})$. Note that, if $\varphi(\vec{x}, \vec{y})$ is Boolean, the set of certain answers is either empty or it consists of a tuple of length zero. We treat unsatisfiability as a Boolean query where $\varphi(\vec{x}, \vec{y})$ is the nullary falsehood symbol $\bot$; this query holds w.r.t. $\mathcal{K}$ iff $\mathcal{K}$ is unsatisfiable.

CQs can be alternatively represented using datalog rules. To this end, each query $q(\vec{x})$ is uniquely associated with a predicate P_q of arity $|\vec{x}|$ (where we take $P_\bot = \bot$) and a set $\mathcal{R}_q$ of rules defined as follows:

$$\mathcal{R}_q = \begin{cases} \emptyset & q = \bot \\ \{\varphi(\vec{x}, \vec{y}) \to P_q(\vec{x})\} & \text{otherwise} \end{cases} \tag{10}$$

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

Answering CQs w.r.t. knowledge bases can be computationally very hard, and decidability for knowledge bases stemming from OWL 2 DL ontologies remains open. The standard language SPARQL 1.1 (W3C SPARQL Working Group, 2013) allows users to formulate CQs over OWL 2 ontologies; however, to ensure decidability and reduce the complexity of query answering, CQs are interpreted in SPARQL 1.1 under *ground semantics*. We say that a possible answer $\vec{a}$ to $q(\vec{x}) = \exists \vec{y} \varphi(\vec{x}, \vec{y})$ is a *ground answer* w.r.t. a satisfiable knowledge base $\mathcal{K}$ if there exists a tuple $\vec{e}$ of constants in $\mathcal{K}$ such that $\mathcal{K} \models \varphi(\vec{a}, \vec{e})$. Clearly, every ground answer is a certain answer but not vice-versa. We denote with $\mathsf{ground}(q, \mathcal{K})$ the set of ground answers to q w.r.t. $\mathcal{K}$.

Many reasoning systems currently support SPARQL 1.1 and hence compute $\mathsf{ground}(q, \mathcal{K})$ when given a CQ q and an OWL 2 DL ontology $\mathcal{K}$ as input. Additionally, most systems are able to compute all certain answers if q is suitably restricted. More precisely, we say that q is *internalisable* if $\mathcal{K}_q = \mathcal{K} \cup \mathcal{R}_q$ corresponds to an OWL 2 DL knowledge base. Internalisation amounts to transforming the query into an ontology axiom and it is typically referred to as *rolling-up* in the DL literature (Horrocks & Tessaris, 2000).

In this paper, we focus on the general problem of computing all certain answers of a CQ w.r.t. a knowledge base $\mathcal{K}$, and all our theoretical results are generally applicable regardless of the rule-based language in which $\mathcal{K}$ is expressed.

2.4 Hyperresolution

Reasoning over knowledge bases can be realised by means of the *hyperresolution calculus*, which we briefly discuss next (Robinson & Voronkov, 2001). In our treatment of hyperresolution we consider standard basic notions in theorem proving such as *(ground) clause* and *most general unifier* (MGU). Furthermore, we treat disjunctions of ground atoms as sets and hence we do not allow for duplicated atoms in a disjunction. We assume that $\bot$ does not occur in clauses and denote with $\Box$ the empty clause. 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.

Let $C = \neg\beta_1 \vee \dots \vee \neg\beta_n \vee \gamma_1 \vee \dots \vee \gamma_m$ be a clause, where each β_i and γ_j are atoms (possibly containing functional terms). Furthermore for each $1 \leq i \leq n$, let $\psi_i = \alpha_i \vee \chi_i$ be a positive ground clause. Finally, let σ be a MGU of all pairs β_i, α_i, $1 \leq i \leq n$. Then, the positive ground clause $\gamma_1\sigma \vee \dots \vee \gamma_m\sigma \vee \chi_1 \vee \dots \vee \chi_n$ is a *hyperresolvent* of C and $\psi_1, \dots, \psi_n$. The inference is called a *hyperresolution step*, where the clause C is the *main premise.*

Let $\mathcal{K} = \Sigma_{\mathcal{K}} \cup \mathcal{D}_{\mathcal{K}}$ be a normalised knowledge base and let C be a positive ground clause. A *derivation* of C 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) $\lambda(v) = C$ if v is the root;

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

(3) if v has children $w_1, \dots, w_n$, then $\lambda(v)$ is a hyperresolvent of $\mathsf{sk}(r)$ and $\lambda(w_1), \dots, \lambda(w_n)$ for a rule $r \in \Sigma_{\mathcal{K}}$.

The *support* of ρ, written $\mathsf{support}(\rho)$, is the set of facts and rules participating in hyperresolution steps in ρ. We write $\mathcal{K} \vdash C$ to denote that there is a hyperresolution derivation of C from $\mathcal{K}$. Hyperresolution is sound and complete: $\mathcal{K}$ is unsatisfiable if and only if $\mathcal{K} \vdash \Box$; furthermore, if $\mathcal{K}$ is satisfiable then $\mathcal{K} \vdash \alpha$ iff $\mathcal{K} \models \alpha$ for any ground atom α.

2.5 The Skolem Chase

Answering CQs over a knowledge base $\mathcal{K} = \Sigma_{\mathcal{K}} \cup \mathcal{D}_{\mathcal{K}}$ where $\Sigma_{\mathcal{K}}$ consists only of existential rules can be realised using the *chase* technique (Abiteboul et al., 1995; Calì, Gottlob, & Kifer, 2013). In this paper, we use the *Skolem chase* variant (Marnette, 2009; Cuenca Grau, Horrocks, Krötzsch, Kupke, Magka, Motik, & Wang, 2013).

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:

$$\mathcal{B}^{i+1} = \mathcal{B}^i \cup \{\mathsf{head}(\mathsf{sk}(r))\sigma \mid r \in \Sigma_{\mathcal{K}}, \sigma \text{ a substitution, and } \mathcal{B}^i \models \mathsf{body}(r)\sigma\}.$$

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

The key property of the Skolem chase is that it computes a universal Herbrand model of $\mathcal{K}$, which can be used as a 'database' for answering CQs. Formally, $\mathcal{K}$ is satisfiable iff $\bot \notin \mathsf{Chase}_{\mathcal{K}}$; furthermore, if $\mathcal{K}$ is satisfiable, then $\mathsf{Chase}_{\mathcal{K}}$ is homomorphically embeddable into every Herbrand model of $\mathcal{K}$ (seen as a set of atoms). It follows that for $\mathcal{K}$ satisfiable and q a Boolean CQ we have $\mathcal{K} \models q$ iff $\mathsf{Chase}_{\mathcal{K}} \models q$.

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

3. Overview

In this section we provide a high-level overview of our approach to conjunctive 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 DL reasoners can typically process arbitrary OWL 2 DL knowledge bases; however, the query language is limited to internalisable queries. 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 this paper, we fix an arbitrary normalised knowledge base $\mathcal{K} = \Sigma_{\mathcal{K}} \cup \mathcal{D}_{\mathcal{K}}$. Given an arbitrary 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 to 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 Sections 4 and 5.

- **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, $\vec{a} \in \mathsf{cert}(q, \mathcal{K})$ iff $\vec{a} \in \mathsf{cert}(q, \mathcal{K}^q)$ for each $\vec{a} \in G^q$. The details on how to compute such $\mathcal{K}^q$ are given in Section 6.

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

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

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

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

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

   i. Construct a summary of $\mathcal{K}^\perp$ (See Section 7), 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 $\mathcal{K}^\perp$; if it is unsatisfiable, then terminate and report that $\mathcal{K}$ is unsatisfiable. Otherwise, proceed to Step 2 ($\mathcal{K}$ is satisfiable).

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

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

$$q_{ex}(x) = \exists y[\mathsf{eats}(x, y) \wedge \mathsf{Plant}(y)]$$

Table 2: Running example knowledge base $\mathcal{K}_{ex}$ and query $q_{ex}(x)$. The set $\Sigma_{\mathcal{K}_{ex}}$ consists of rules (R1)–(R9), the dataset $\mathcal{D}_{\mathcal{K}_{ex}}$ consists of the facts (D1)–(D15).

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 $\mathcal{K}$.

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

(a) Construct a summary of $\mathcal{K}^q$ (see Section 7). For each $\vec{a} \in G_q$, use the fully-fledged reasoner to check whether the summary of $\vec{a}$ is a certain answer to q w.r.t. the summary of $\mathcal{K}^q$, and remove $\vec{a}$ from G^q if it is not the case.

(b) Compute a dependency relation between the remaining answers in G^q s.t. if $\vec{b}$ depends on $\vec{a}$ and $\vec{a}$ is a spurious answer, then so is $\vec{b}$. (See Section 7).

(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 it depends on a spurious answer; 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 sections, 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 $\mathcal{K}'$ and query q', it computes query answers $\mathsf{cert}(q', \mathcal{K}')$ by first computing the materialisation $\mathsf{Chase}_{\mathcal{K}'}$ 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. Our PAGOdA system combines HermiT with the materialisation-based reasoner RDFox, and hence is able to exploit all of the improvements and optimisations described below; the realisation of our approach in PAGOdA is discussed in detail in Section 8.

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 $q_{ex}(x)$ given in Table 2. 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 follows:

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

Our core techniques described in Sections 4-6 are applicable to any knowledge base and 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 $q(\vec{x}) = \exists \vec{y} \varphi(\vec{x}, \vec{y})$ (which may be the unsatisfiability query $\bot$).

4. 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 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 4.1. The datalog subset of our example $\mathcal{K}_{ex}$ consists of rules (R1)–(R4) and (R9), together with all facts (D1)–(D15). The materialisation of the datalog subset of $\mathcal{K}_{ex}$ results in the following dataset: $\mathcal{D}_{ex} \cup \{\mathsf{Mammal}(rabbit), \mathsf{Mammal}(sheep), \mathsf{Plant}(grass)\}$ When evaluating $q_{ex}(x)$ against the materialisation we obtain *sheep* as an answer. $\diamondsuit$

This basic lower bound can be rather imprecise in practice since rules featuring disjunction or existential quantification typically abound in OWL 2 DL 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.

4.1 Dealing with Disjunctive Rules: Program Shifting

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

Example 4.2. Let us consider the information in $\mathcal{K}_{ex}$ about Arctic hares (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 $willow$, 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 then $\mathsf{Herbivore}(a\_hare)$. ♢

We now define the shifting transformation formally.

Definition 4.3. Let r be a normalised disjunctive datalog rule. For each predicate P in r let $\overline{P}$ be a fresh predicate of the same arity. Furthermore, given an atom $\alpha = P(\vec{t})$ let $\overline{\alpha}$ be $\overline{P}(\vec{t})$. The shifting of r, written $\mathsf{shift}(r)$, is the following set of rules:

- if r of the form (2), 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 (4), 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 $1 \leq i \leq n$ s.t. 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:

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

♢

Note that shifting is a polynomial transformation. For r a disjunctive datalog rule 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 4.4. *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 $\mathsf{Chase}_{\mathcal{K}'} = \{\mathcal{B}^i\}_{i=1}^{L}$ with L some non-negative integer (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 $\alpha = P(\vec{a})$, then $\mathcal{K} \models P(\vec{a})$; and

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

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 $\beta_1 \wedge \dots \wedge \beta_n \to \gamma_1 \vee \dots \vee \gamma_m$ 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, $\mathcal{K} \models \{\beta_1\sigma, \dots, \beta_n\sigma\}$ and hence $\mathcal{K} \models \bot$; *(ii)* $\mathsf{head}(r) \neq \bot$, in which case r' is of the form (S1) and $\beta_1\sigma, \dots, \beta_n\sigma$ and $\overline{\gamma}_1\sigma, \dots, \overline{\gamma}_m\sigma$ are in $\mathcal{B}^i$. By the induction hypothesis, $\mathcal{K}$ entails $\beta_1\sigma, \dots, \beta_n\sigma$ and $\neg\gamma_1\sigma, \dots, \neg\gamma_m\sigma$. But then, rule r cannot be satisfied by any model of $\mathcal{K}$ and since $r \in \mathcal{K}$, we obtain that $\mathcal{K}$ is unsatisfiable.

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

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

If $q = \bot$, the theorem follows from property (a). Otherwise, let $q(\vec{x}) = \exists\vec{y}(\bigwedge_{i=1}^n \beta_i(\vec{x}, \vec{y}))$ and let $\vec{a}$ be a possible answer such that $\mathcal{K}' \models q(\vec{a})$. Since $\mathcal{K}'$ is datalog, there exists a tuple $\vec{e}$ of constants in $\mathcal{K}'$ and a non-negative integer L such that $\beta_i(\vec{a}, \vec{e}) \in \mathcal{B}^L$ for each $0 \leq i \leq n$. But then, by (b) we have $\mathcal{K} \models \beta_i(\vec{a}, \vec{e})$, and hence $\mathcal{K} \models q(\vec{a})$. □

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

Example 4.5. Consider the disjunctive datalog knowledge base consisting of the fact $\mathsf{GreenSeaTurtle}(turtle)$, the rules (R1), (R2) and

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

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

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

It can be easily checked that fact Mammal(*turtle*) does not follow from the shifting. ◇

4.2 Dealing with Existential Rules: The Combined Approach for $\mathcal{ELHO}^r_\bot$

To compute query answers that depend on existentially quantified rules, we consider the subset of $\mathcal{ELHO}^r_\bot$ rules and facts in $\mathcal{K}$. In the case of $\mathcal{ELHO}^r_\bot$ knowledge bases, 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). The combined approach can be conceptualised as a three-step process.

1. The first step is to compute the materialisation M of a datalog program obtained from 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. This model, however, is not universal as it cannot be homomorphically embedded into every other model. Thus, the evaluation of CQs over M may lead to unsound answers.

2. The second step is to evaluate the query q over M. This step is intractable in query size, but well-known database techniques can be exploited.

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 knowledge bases to datalog used in the first step, as this transformation will also be exploited later on in Section 5 for computing upper bound query answers. Describing the technical details of filtration, however, is beyond the scope of this paper, and we refer the reader to the relevant literature (Stefanoni et al., 2013). For our purposes, it suffices to assume the availability of a procedure soundAnswers that solves Steps 2 and 3; that is, given q and the model computed in Step 1, it returns all answers to q w.r.t. the input $\mathcal{ELHO}^r_\bot$ knowledge base.

The computation of the datalog program from a 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 4.6. For each rule r of the form (1) and each existentially quantified variable z_{ij}, let c^r_{ij} be a constant globally unique for r and z_{ij}, and let $\theta_{\mathsf{c\text{-}sk}}$ be the substitution such that $\theta_{\mathsf{c\text{-}sk}}(z_{ij}) = \mathsf{c}^r_{ij}$ for each $z_{ij} \in \vec{z}_i$. The *c-Skolemisation* $\mathsf{c\text{-}sk}(r)$ of r is given as follows:

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

Then, we define $\mathsf{c\text{-}sk}(\mathcal{K}) = \{\mathsf{c\text{-}sk}(r) \mid r \in \Sigma_\mathcal{K}\} \cup \mathcal{D}_\mathcal{K}$. ◇

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

- extract the subset $\Sigma^{EL}_\mathcal{K}$ of all $\mathcal{ELHO}^r_\bot$ rules in $\mathcal{K}$;
- 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 $\mathsf{soundAnswers}(q, M)$.

Example 4.7. 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

$$\mathsf{Folivore}(x) \to \mathsf{eats}_L(x, \mathsf{c}_3) \tag{R8aU}$$

The materialisation of the datalog program consisting of all facts and rule (R8aU) contains the fact $\mathsf{Plant}(c_3)$ and hence the tuple $(howler, c_3)$ matches q_{ex} to the materialisation. This match is deemed sound by the filtration procedure. ♢

4.3 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. Construct the datalog knowledge base $\mathsf{shift}(\Sigma^{DD}_\mathcal{K}) \cup \mathcal{D}_\mathcal{K}$, where $\Sigma^{DD}_\mathcal{K}$ is the subset of disjunctive datalog rules in $\Sigma_\mathcal{K}$. Compute its materialisation M^L_1.
2. Construct the datalog program $\mathsf{c\text{-}sk}(\Sigma^{EL}_\mathcal{K}) \cup M^L_1$ and compute its materialisation M^L_2.
3. If $q = \bot$, then $L^q = \mathsf{cert}(q, M^L_2)$. Otherwise, $L^q = \mathsf{soundAnswers}(q, M^L_2)$.

Theorem 4.4 ensures that $\mathcal{K} \models \alpha$ for any $\alpha \in M^L_1$ in the signature of $\mathcal{K}$, and hence M^L_1 can be used as the initial dataset for the second step. The properties of c-Skolemisation and filtration discussed in Section 4.2 then ensure that every answer in L^q is indeed a certain answer of q w.r.t. $\mathcal{K}$. Furthermore, if $\bot \in M^L_2$, then $\mathcal{K}$ is indeed unsatisfiable. Finally, note that the materialisation M^L_1 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. $\mathsf{shift}(\Sigma^{DD}_\mathcal{K}) \cup \mathcal{D}_\mathcal{K}$ and $\mathsf{c\text{-}sk}(\Sigma^{EL}_\mathcal{K}) \cup \mathcal{D}_\mathcal{K}$ independently and then unioning the results.

Example 4.8. For our running example $\mathcal{K}_{ex}$, the aggregated lower bound L_{ex} consists of *sheep* (which follows from the datalog subset of $\mathcal{K}_{ex}$), *a_hare* (which follows from $\mathsf{shift}(\mathcal{K}_{ex})$), and *howler* (which follows from the $\mathcal{ELHO}^r_\bot$ fragment of $\mathcal{K}_{ex}$). ♢

5. Upper Bound Computation

In many practical cases the lower bound L^q described in Section 4.3 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 section, we describe our approach to efficiently computing an upper bound to the set of certain answers. If lower and upper bounds coincide, then we have fully answered the query; 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.

5.1 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 occurring in head position into different datalog rules; and
3. Skolemising existentially quantified variables into constants as in Definition 4.6.

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. The datalog strenghtening 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 5.1. The *splitting* of a rule r of the form (1) is the following set of rules:

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

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 strenghtening* of $\mathcal{K}$ is defined as $\mathsf{str}(\mathcal{K}) = \mathsf{c\text{-}sk}(\mathsf{split}(\mathcal{K}))$. ♢

Example 5.2. 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).

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

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

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

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

$$\mathsf{MeatEater}(x) \rightarrow \mathsf{eats}_H(x, \mathsf{c}_1) \tag{R6aU}$$
$$\mathsf{Mammal}(x) \rightarrow \mathsf{eats}(x, \mathsf{c}_2) \tag{R7U}$$

as well as rule (R8a) with rule (R8aU) given in Example 4.7. $\diamondsuit$

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 $\mathsf{cert}(q, \mathcal{K})$. This is, however, no longer the case if $\bot$ is replaced with an ordinary predicate $\bot_s$ without a predefined meaning. 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 streghtened knowledge base unsatisfiable.

Example 5.3. Consider the strenghtening $\mathcal{K}'_{ex} = \mathsf{str}(\mathcal{K}_{ex})$ 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 strenghtening of $\mathcal{K}_{ex}$ would be unsatisfiable, in which case no meaningful upper bound could be obtained for any query. $\diamondsuit$

We next show that $\mathsf{str}(\mathcal{K})$ 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 5.1: 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 5.4. *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 $\alpha \in \mathsf{Chase}_{\mathcal{H}}$.*

Proof. We prove the claim by structural induction on ρ.

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 $\alpha \in \mathsf{Chase}_{\mathcal{H}}$.

Inductive step: Assume that the induction hypothesis holds for all children $w_1, \ldots, w_n$ of a node $v \in T$. There exists a rule $r \in \Sigma_{\mathcal{K}}$ and a substitution σ, where $\mathsf{sk}(r)$ is of the form $\neg\beta_1 \vee \cdots \vee \neg\beta_n \vee \psi$ with ψ a disjunction of atoms, such that $\lambda(v) = \psi\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is the hyperresolvent of $\mathsf{sk}(r)$ and $\lambda(w_i) = \beta_i\sigma \vee \chi_i$ for each $1 \leq i \leq n$. By the induction hypothesis, all the disjuncts in each χ_i are in $\mathsf{Chase}_{\mathcal{H}}$, 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 (2), $\psi\sigma$ is empty. So the claim holds vacuously.
- If r is of the form (3), then $\psi = \gamma_1\theta_{\mathsf{sk}}$. By the induction hypothesis, each $\beta_i\sigma$ is in $\mathsf{Chase}_{\mathcal{H}}$, and since $\mathsf{split}(r) = r$ and hence $r \in \mathcal{H}$, we obtain $\gamma_1\theta_{\mathsf{sk}}\sigma \in \mathsf{Chase}_{\mathcal{H}}$.

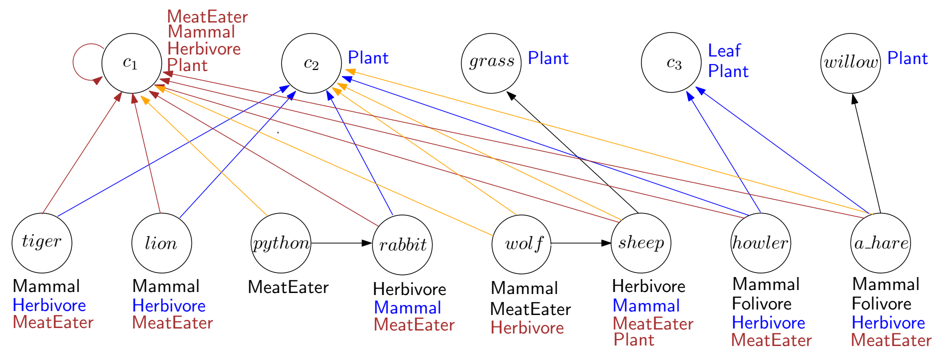


Figure 1: Materialisation of Datalog strengthening of $\mathcal{K}_{ex}$

- If r is of the form (4), then $\psi = \gamma_1 \vee \cdots \vee \gamma_m$. By induction hypothesis, each $\beta_i\sigma$ is in $\mathsf{Chase}_{\mathcal{H}}$, and for each $1 \leq i \leq m$, since the rule $\beta_1 \wedge \cdots \wedge \beta_n \to \gamma_i$ is in $\mathcal{H}$, we obtain that each atom $\gamma_i\sigma$ is also in $\mathsf{Chase}_{\mathcal{H}}$, 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 may be exploited to compute upper bound query answers.

Theorem 5.5. *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 (2) 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 5.4 we have that $\beta_i\sigma \in \mathsf{Chase}_{\mathcal{H}}$. Since $\mathcal{H}$ contains the rule $\beta_1 \wedge \cdots \wedge \beta_n \to \bot_s$ we have $\bot_s \in \mathsf{Chase}_{\mathcal{H}}$ 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 $\vec{a}$ be a certain answer to q w.r.t. $\mathcal{K}$. So $\mathcal{K} \models q(\vec{a})$ and hence $\mathcal{K} \cup \mathcal{R}_q \models P_q(\vec{a})$. Since $\mathsf{cert}(\bot, \mathcal{K}) = \emptyset$, we have $q \neq \bot$. Using the completeness of hyperresolution and Lemma 5.4 we obtain that $P_q(\vec{a})$ is in the chase of $\mathcal{K} \cup \mathcal{R}_q$. But then, the aforementioned splitting also entails $P_q(\vec{a})$ and since $\mathsf{split}(\mathcal{K} \cup \mathcal{R}_q) = \mathcal{H} \cup \mathcal{R}_q$ we have $\vec{a} \in \mathsf{cert}(q, \mathcal{H})$, 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 5.6. Figure 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 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:

$$\mathsf{cert}(q_{ex}, \mathsf{str}(\mathcal{K}_{ex})) = \{tiger, lion, python, rabbit, wolf, sheep, howler, a\_hare, \mathsf{c}_1\}$$

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 c_1 is a certain answers to q_{ex} w.r.t. $\mathsf{str}(\mathcal{K}_{ex})$; however, constant c_1 is not in the signature of $\mathcal{K}_{ex}$ and hence it is not a possible answer to q_{ex} w.r.t. $\mathcal{K}$. $\diamondsuit$

5.2 Tightening the Upper Bound: Existential Rules

The upper bound obtained from $\mathsf{str}(\mathcal{K})$ can be rather coarse-grained in practice: as discussed in Example 5.6, *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 and then applying 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 the head atom (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 $\mathsf{split}(\mathcal{K})$.

2. 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 5.7. Consider again the materialisation of $\mathsf{str}(\mathcal{K}_{ex})$ in Figure 1. As already mentioned, *python* is returned as an upper bound answer since q_{ex} matches the facts $\mathsf{eats}(\mathit{python}, \mathsf{c}_1)$ and $\mathsf{Plant}(\mathsf{c}_1)$ in the materialisation. The fact $\mathsf{eats}(\mathit{python}, \mathsf{c}_1)$ is obtained from $\mathsf{eats}_H(\mathit{python}, \mathsf{c}_1)$, 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 $\mathsf{eats}_H(\mathit{python}, \mathit{rabbit})$, which is derived from $\mathsf{eats}(\mathit{python}, \mathit{rabbit})$ and $\mathsf{Herbivore}(\mathit{rabbit})$ in the dataset, and rule (R6b). Please note that rule (R6b) is of the form (9) 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 *python* during the chase, and dispense with $\mathsf{eats}(\mathit{python}, \mathsf{c}_1)$ in the materialisation. $\diamondsuit$

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

Definition 5.8. 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 sets of ground atoms $\{\mathcal{B}^i\}_{i \geq 0}$, 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

$\mathcal{S}_e^{i+1}$ be defined as follows:

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

Then, $\mathcal{B}^{i+1} = \mathcal{B}^i \cup \mathcal{S}_d^{i+1}$ if $\mathcal{S}_d^{i+1} \neq \emptyset$, and $\mathcal{B}^{i+1} = \mathcal{B}^i \cup \mathcal{S}_e^{i+1}$ 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 $\mathsf{c\text{-}sk}(\mathsf{split}(\mathcal{K}))$.

Example 5.9. The c-chase of $\mathcal{K}_{ex}$ is depicted in Figure 2. This materialisation is a strict subset of that in Figure 1, where the orange-coloured binary facts 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 5.10. *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 (3). 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 5.5. 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 $\vec{a}$ be a certain answer to q w.r.t. $\mathcal{K}$. By Theorem 5.5, we obtain $\vec{a} \in \mathsf{cert}(q, \mathsf{split}(\mathcal{K}))$. Because $\mathsf{c\text{-}Chase}_{\mathcal{K}} \models \mathsf{split}(\mathcal{K})$, we have $\vec{a} \in \mathsf{cert}(q, \mathsf{c\text{-}Chase}_{\mathcal{K}})$. □

5.3 Tightening the Upper Bound: 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 (4) 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.*

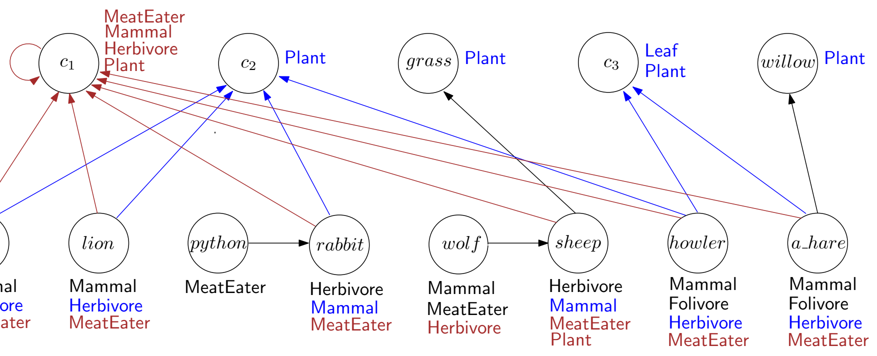


Figure 2: c-chase of $\mathcal{K}_{ex}$

Example 5.11. 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 2. Indeed, $\mathsf{Herbivore}(\mathit{wolf})$ is derived from $\mathsf{Mammal}(\mathit{wolf})$ and the rules in the split of (R5); thus, $\mathsf{Plant}(\mathit{sheep})$ is also derived using rule (R4). Note, however, that *wolf* is a spurious answer: given that $\mathsf{MeatEater}(\mathit{wolf})$ is an explicit fact in $\mathcal{K}_{ex}$, rule (R5) is already satisfied for *wolf* and hence we can dispense with fact $\mathsf{Herbivore}(\mathit{wolf})$ 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. ♢

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

Definition 5.12. 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 ground clause χ and a set of ground atoms returns a disjunct in χ. The *c-chase* sequence of $\mathcal{K}$ w.r.t. f is the sequence of sets of ground atoms $\{\mathcal{B}^i\}_{i\geq 0}$, 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:

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

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 $\textsf{c-Chase}^f_{\mathcal{K}} = \bigcup_{i\geq 0}\{\mathcal{B}^i\}$. ♢

Example 5.13. Consider the aforementioned choice function f that picks $\mathsf{Herbivore}(b)$ whenever rule (R5) is applied to a fact $\mathsf{Mammal}(b)$. Figure 3 depicts the facts in $\textsf{c-Chase}^f_{\mathcal{K}_{ex}}$. It can be observed that $\textsf{c-Chase}^f_{\mathcal{K}_{ex}}$ is a strict subset of the materialisation in Figure 2, where the brown-colored facts are no longer derived. We can see that *wolf* is not an answer to q_{ex} w.r.t. $\textsf{c-Chase}^f_{\mathcal{K}_{ex}}$ 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. ♢

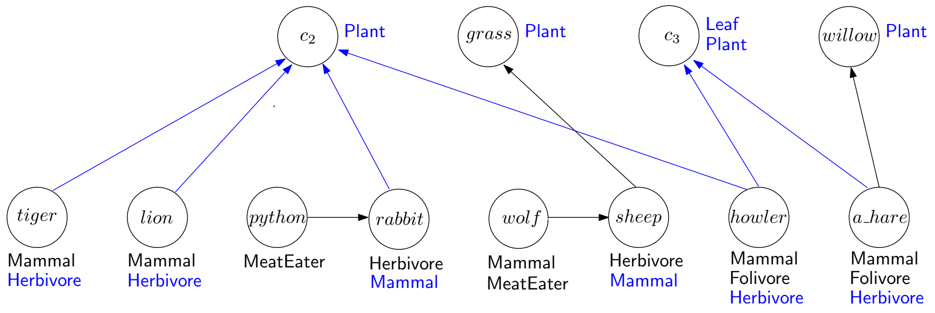


Figure 3: c-chase^f of $\mathcal{K}_{ex}$

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

Theorem 5.14. *Let f be a choice function as in Definition 5.12. 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 (2). 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 (3). Pick 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 (4). Pick 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 from the fact that $\mathsf{c\text{-}Chase}^f_{\mathcal{K}}$ is a model of $\mathcal{K}$. □

5.4 Combined Upper Bound

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

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. $\mathcal{K}$ (c.f. Section 5.1).

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. $\mathcal{K}$ (c.f. Section 5.2).

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. $\mathcal{K}$ (c.f. Section 5.3).

It can be trivially seen that U^q_2 and U^q_3 are more precise than 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 5.15. 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 $B(x) \to C(x) \vee D(x)$, $R(x, y) \wedge C(y) \to S(x, y)$ and $A(x) \to \exists y S(x, y)$. Let c be the freshly introduced constant for $A(x) \to \exists y S(x, y)$, 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,

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

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^q_1 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^q_3 has an important shortcoming: it is of no use whenever $\bot_s$ is derived, as we will show in the following example.

Example 5.16. 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 combine U^q_2 and U^q_3 to compute a hopefully more precise upper bound; otherwise, we can use U^q_2. The combined upper bound query answer U^q to q in $\mathcal{K}$ is formally defined as follows:

$$U^q = \begin{cases} U^{\bot_s}_2 \cap U^{\bot_s}_3 & \text{if } q = \bot; \\ U^q_2 \cap U^q_3 & \text{if } q \neq \bot \text{ and } \bot_s \notin \mathsf{c\text{-}Chase}^f_{\mathcal{K}}; \\ U^q_2 & \text{otherwise.} \end{cases} \tag{13}$$

Example 5.17. The combined upper bound for q_{ex} in $\mathcal{K}_{ex}$ gives:

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

If we compare this upper bound with the aggregated lower bound L_{ex} given in Example 4.8 we can identify a gap $G_{ex} = \{\mathit{tiger}, \mathit{lion}, \mathit{rabbit}\}$. ♢

6. 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 a 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. $\vec{a} \in \mathsf{cert}(q, \mathcal{K})$ iff $\vec{a} \in \mathsf{cert}(q, \mathcal{K}^q)$ for each $\vec{a} \in G^q$). 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 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 $\Box$ (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.

6.1 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 5.13). 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:

$$\mathsf{Plant}^R(y) \wedge \mathsf{Herbivore}(x) \wedge \mathsf{eats}(x,y) \rightarrow \mathsf{Herbivore}^R(x) \tag{14}$$

$$\mathsf{Plant}^R(y) \wedge \mathsf{Herbivore}(x) \wedge \mathsf{eats}(x,y) \rightarrow \mathsf{eats}^R(x,y) \tag{15}$$

$$\mathsf{Plant}^R(y) \wedge \mathsf{Herbivore}(x) \wedge \mathsf{eats}(x,y) \rightarrow \mathsf{Rel}(\mathsf{d}_{R4}) \tag{16}$$

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 facts $\mathsf{Herbivore}(b)$ and $\mathsf{eats}(b,c)$ 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 (14)–(16) faithfully 'invert' hyperresolution steps involving rule (R4).

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

$$\mathsf{Herbivore}^R(x) \wedge \mathsf{MeatEater}^R(x) \wedge \mathsf{Mammal}(x) \rightarrow \mathsf{Mammal}^R(x) \tag{17}$$

$$\mathsf{Herbivore}^R(x) \wedge \mathsf{MeatEater}^R(x) \wedge \mathsf{Mammal}(x) \rightarrow \mathsf{Rel}(\mathsf{d}_{R5}) \tag{18}$$

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 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 and 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:

$$\mathsf{eats}^R(x,y) \wedge \mathsf{Mammal}(x) \rightarrow \mathsf{Mammal}^R(x) \tag{19}$$

$$\mathsf{eats}^R(x,y) \wedge \mathsf{Mammal}(x) \rightarrow \mathsf{Rel}(\mathsf{d}_{R7}) \tag{20}$$

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

$$P^R_{q_{ex}}(x) \wedge \mathsf{eats}(x,y) \wedge \mathsf{Plant}(y) \rightarrow \mathsf{eats}^R(x,y) \tag{21a}$$

$$P^R_{q_{ex}}(x) \wedge \mathsf{eats}(x,y) \wedge \mathsf{Plant}(y) \rightarrow \mathsf{Plant}^R(y) \tag{21b}$$

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 (21a) and (21b) 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.

6.2 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 6.1. 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 $\alpha = P(\vec{t})$, let α^R denote $P^R(\vec{t})$. For any normalised rule $r \in \mathcal{K} \cup \mathcal{R}_q$, let $\mathsf{move}(r)$ be the following conjunction of atoms:

- $P^R_\bot$ if r of the form (2);
- $\gamma^R_1(\vec{x}, \vec{z}_1)$ if r of the form (3); and
- $\gamma^R_1(\vec{x}) \wedge \dots \wedge \gamma^R_m(\vec{x})$ if r of the form (4).

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

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

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 $P^R_q(\vec{a})$ for each $\vec{a} \in G$; and

(iv) a fact $P^R_\bot$ 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 Sections 4.3 and 5.4. ♢

Note that $\mathcal{K}^{\perp}$ is a subset of $\mathcal{K}^{q}$ since $\mathsf{track}(\mathcal{K}, \perp, G^{\perp})$ is a subset of $\mathsf{track}(\mathcal{K}, q, G^{q})$: in Definition 6.1, point *(i)* is the same for $\mathsf{track}(\mathcal{K}, \perp, G^{\perp})$ and $\mathsf{track}(\mathcal{K}, q, G^{q})$; furthermore, the set of rules from *(ii)* for $\mathsf{track}(\mathcal{K}, \perp, G^{\perp})$ is a subset of that for $\mathsf{track}(\mathcal{K}, q, G^{q})$ since $\mathcal{K} \cup \mathcal{R}_{\perp} \subseteq \mathcal{K} \cup \mathcal{R}_{q}$; finally, the fact $P_{\perp}^{R}$, which is included in $\mathsf{track}(\mathcal{K}, \perp, G^{\perp})$ by point *(iii)*, also belongs to $\mathsf{track}(\mathcal{K}, q, G^{q})$ by point *(iv)*.

Example 6.2. 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. ♢

The key properties of the computed subsets are established by the following theorem.

Theorem 6.3. *The following properties hold:*

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

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

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 any clause in $\Upsilon(q, G)$ from $\mathcal{K} \cup \mathcal{R}_{q}$ where

$$\Upsilon(q, G) = \begin{cases} \{\Box\} & \text{if } q = \perp; \\ \{P_{q}(\vec{a}) \mid \vec{a} \in G\} & \text{otherwise.} \end{cases}$$

Claim (♣) If $\rho = (\lambda, T)$ is a derivation of $\alpha \in \Upsilon(q, G)$ 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 5.10, Theorem 5.14 and (13), we have $U^{\perp} \neq \emptyset$ and thus $G^{\perp} \neq \emptyset$. There exists a hyperresolution derivation ρ_1 of $\Box$ from $\mathcal{K}$. Since $\Upsilon(\perp, G^{\perp}) = \{\Box\}$, we know that $\mathsf{support}(\rho_1) \subseteq \mathcal{K}^{\perp}$ by (♣). So $\mathcal{K}^{\perp}$ is unsatisfiable. To show the 'only if' direction of *(2)*, assume that $\vec{a} \in G$ and $\vec{a} \in \mathsf{cert}(q, \mathcal{K})$. Then there exists a hyperresolution ρ_2 of $P_{q}(\vec{a})$ from $\mathcal{K} \cup \mathcal{R}_{q}$. Similarly, by (♣), we know that $\mathsf{support}(\rho_2) \subseteq \mathcal{K}^{q,G}$ and hence $\vec{a} \in \mathsf{cert}(q, \mathcal{K}^{q,G})$.

We next show inductively a statement from which (♣) will follow. Let $\rho = (\lambda, T)$ be a derivation of a clause 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 5.10) that $\mathsf{c\text{-}Chase}_{\mathcal{K}}$ is a model of $\mathcal{H}$. Since $\mathsf{Chase}_{\mathcal{H}}$ is a universal model of $\mathcal{H}$ there exists a homomorphism τ from $\mathsf{Chase}_{\mathcal{H}}$ 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 the parent u of v.

We proceed by induction on the distance of v to the root 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, $\lambda(v) = P_q(\vec{a})$ for some $\vec{a} \in G$. 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. An easy observation of composition between a substitution and a homomorphism is used later in the rest of the proof.

$$(\beta\sigma)\tau = \beta(\sigma\tau) \text{ for an arbitrary function-free atom } \beta. \tag{22}$$

By Lemma 5.4 in Section 5.1 we have that each $\beta_i\sigma \in \mathsf{Chase}_{\mathcal{H}}$ for each $1 \leq i \leq n$. Since τ is a homomorphism from $\mathsf{Chase}_{\mathcal{H}}$ into $\mathsf{c\text{-}Chase}_{\mathcal{K}}$ we then have that $(\beta_i\sigma)\tau \in \mathsf{c\text{-}Chase}_{\mathcal{K}}$ and by (22), $\beta_i(\sigma\tau) \in \mathsf{c\text{-}Chase}_{\mathcal{K}}$ 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 $\mathsf{move}(r) = P_{\bot}^R$. We distinguish two cases.
  - if $q \neq \bot$, $P_{\bot}^R \in \mathsf{track}(\mathcal{K}, q, G)$ by point *(iv)*;
  - if $q = \bot$, we have $\bot_s \in \mathsf{c\text{-}Chase}_{\mathcal{K}}$ and hence $P_q^R \in \mathsf{track}(\mathcal{K}, q, G)$ 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 (22), $\mathsf{track}(\mathcal{K}, q, G) \models \gamma_j^R(\sigma\tau)$ 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 $\mathsf{track}(\mathcal{K}, q, G) \models \beta_i^R(\sigma\tau)$ for $1 \leq i \leq n$. Again by (22), $\mathsf{track}(\mathcal{K}, q, G) \models (\beta_i^R\sigma)\tau$ for $1 \leq i \leq n$. In addition, by the induction hypothesis, we have $\mathsf{track}(\mathcal{K}, q, G) \models \chi_i^R\tau$, 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. $\square$

We conclude with an example illustrating why the dataset in $\mathsf{track}(\mathcal{K}, q, G)$ (point 1 in Definition 6.1) is obtained from $\mathsf{c\text{-}Chase}_{\mathcal{K}}$ the materialisation underpinning the upper bound in Section 5.2, rather than $\mathsf{c\text{-}Chase}_{\mathcal{K}}^f$ in Section 5.3.

Example 6.4. Consider the query $q(x) = E(x)$ and the knowledge base $\mathcal{K}$ consisting of the following rules and facts.

$$\begin{array}{ll} A(x) \rightarrow B(x) \vee D(x) & \qquad D(x) \rightarrow E(x) \\ B(x) \rightarrow E(x) & \qquad A(a) \end{array}$$

Let f be a function always choosing $B(a)$ over $D(a)$, then $\mathsf{c\text{-}Chase}^f_{\mathcal{K}} = \{A(a), B(a), E(a)\}$ and constant a is an answer to $q(x)$ in the gap between lower and upper bound. Suppose that we were to define $\mathsf{track}(\mathcal{K}, q, G)$ as in Definition 6.1 but replacing the facts in point *(i)* with those in $\mathsf{c\text{-}Chase}^f_{\mathcal{K}}$. Since $D(a)$ does not hold in $\mathsf{c\text{-}Chase}^f_{\mathcal{K}}$ the corresponding subset will not contain the rule $D(x) \rightarrow E(x)$, which is essential to derive $E(a)$. ♢

6.3 Optimisations of the Datalog Encoding

To conclude this section, we present two optimisations of the datalog encoding in Definition 6.1 that we will exploit in our system PAGOdA.

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 (17) 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 (17) would now be written as follows:

$$\begin{aligned} &\mathsf{Herbivore}^I(x) \wedge \mathsf{MeatEater}^I(x) \wedge \mathsf{Herbivore}^R(x) \\ &\qquad\qquad\qquad \wedge \mathsf{MeatEater}^R(x) \wedge \mathsf{Mammal}(x) \rightarrow \mathsf{Mammal}^R(x) \end{aligned}$$

Formally, Definition 6.1 can be optimised as given next.

Definition 6.5. Let $\mathcal{K}$, q, G and predicates P^R be as in Definition 6.1. 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 (2);
- $\gamma^I_1(\vec{x}, \vec{z}_1) \wedge \gamma^R_1(\vec{x}, \vec{z}_1)$ if r of the form (3); and
- $\gamma^I_1(\vec{x}) \wedge \cdots \wedge \gamma^I_m(\vec{x}) \wedge \gamma^R_1(\vec{x}) \wedge \cdots \wedge \gamma^R_m(\vec{x})$ if r of the form (4).

Then, $\Delta(r)$ is as in Definition 6.1, and $\mathsf{track}(\mathcal{K}, q, G)$ is also as in Definition 6.1, but extended with the addition of a fact $P^I(\vec{a})$ for each fact $P(\vec{a})$ that is in $\mathsf{c\text{-}Chase}_{\mathcal{K}}$ but not in $\mathcal{D}_{\mathcal{K}}$. $\diamondsuit$

It is easy to see that this optimisation does not affect the correctness of Theorem 6.3: 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.

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 (EQ4) are inverted into the following rules in $\Delta(r)$ for each predicate P:

$$P^R(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) \wedge P(x_1, \ldots, x_n) \wedge x_i \approx y \rightarrow P^R(x_1, \ldots, x_n) \tag{23}$$

$$P^R(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) \wedge P(x_1, \ldots, x_n) \wedge x_i \approx y \rightarrow \approx^R (x_i, y) \tag{24}$$

where (23) is an tautology and can be dispensed with, but rule (24) 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 (EQ1), (EQ2) or (EQ3), and we only need to include rules (24) 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 6.6. Consider a knowledge base $\mathcal{K}$ consists of facts $\{R(a_1, b), R(a_2, b), A(a_1)\}$ and the following rules.

$$A(x) \rightarrow B(x) \vee C(x) \quad (25) \qquad\qquad B(x) \rightarrow D(x) \quad (27)$$

$$R(x_1, y) \wedge R(x_2, y) \rightarrow x_1 \approx x_2 \quad (26) \qquad\qquad C(x) \rightarrow D(x) \quad (28)$$

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 (26) 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 (24). $\diamondsuit$

7. Summarisation and Analysis of Answer Dependencies

In this section, 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. We therefore 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 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 $\mathcal{K}^{\perp}$ and check its satisfiability. If it is satisfiable, then $\mathcal{K}^{\perp}$ (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 the summary of $\vec{a}$ is entailed to be a certain answer of q in the summary of $\mathcal{K}^q$, discarding any answers that are not so entailed.

Formally, summarisation is defined as follows.

Definition 7.1. 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 σ mapping each constant c in $\mathcal{K}$ to a_T, where T is the type for c. Finally, the summary of $\mathcal{K}$ is $\mathcal{K}\sigma$. $\diamondsuit$

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 $\mathcal{K}^{\perp}$ and $\mathcal{K}^q$ are already significantly smaller than $\mathcal{K}$, checking the satisfiability of $\mathcal{K}^{\perp}$ 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 7.2. *Let σ be the summary function over $\mathcal{K}$. Satisfiability of $\mathcal{K}^{\perp}\sigma$ implies the following: (i) $\mathcal{K}$ is satisfiable; and (ii) $\mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q\sigma, \mathcal{K}^q\sigma)$ for every CQ q.*

Example 7.3. In the case of our running example, the 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 $\{\mathsf{Mammal}\}$. 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. $\diamondsuit$

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 $\vec{a}$ depends on answer $\vec{c}$, and $\vec{c}$ is spurious, then so is $\vec{a}$.

Consider two tuples $\vec{c}$ and $\vec{d}$ of constants in G^q. Suppose that we can find an endomorphism η of the dataset $\mathcal{D}_{\mathcal{K}}$ in which $\vec{c}\eta = \vec{d}$. If we can determine (by calling the fully-fledged reasoner) that $\vec{d}$ is a spurious answer, then so must be $\vec{c}$; as a result, we no longer need to call the fully-fledged reasoner to verify $\vec{c}$. Such endomorphisms are defined next.

Definition 7.4. Let $\vec{c} = (c_1, \dots, c_n)$ and $\vec{d} = (d_1, \dots, d_n)$ be n-tuples of constants from $\mathcal{K}$. An *endomorphism* from $\vec{c}$ to $\vec{d}$ 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)* $P(t_1, \dots, t_m)\eta \in \mathcal{D}_{\mathcal{K}}$ for each fact $P(t_1, \dots, t_m) \in \mathcal{D}_{\mathcal{K}}$; and *(iii)* $r\eta \in \Sigma_{\mathcal{K}}$ for each $r \in \Sigma_{\mathcal{K}}$. $\diamondsuit$

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

Proposition 7.5. *Let $\vec{c}$, $\vec{d}$ be possible answers to q and let η be an endomorphism from $\vec{c}$ to $\vec{d}$ in $\mathcal{K}$. Then, $\vec{c} \in \mathsf{cert}(q, \mathcal{K})$ implies $\vec{d} \in \mathsf{cert}(q, \mathcal{K})$.*

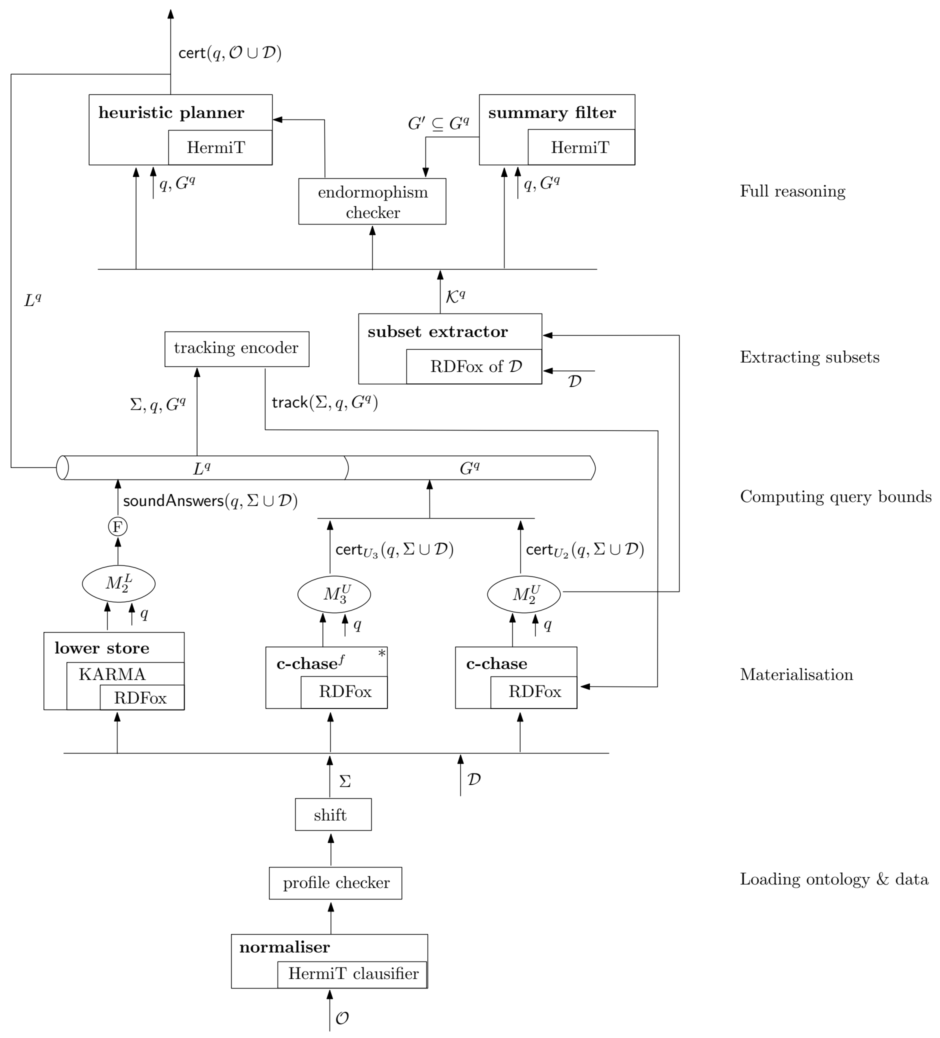


Figure 4: The architecture of PAGOdA

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

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

8. Implementation: The PAGOdA System

We have implemented our approach in a system called PAGOdA, which is written in Java and it is available under an academic license. Our system integrates the datalog reasoner RDFox (Motik et al., 2014) and the fully-fledged OWL 2 reasoner HermiT (Glimm et al.,

2014) as 'black-boxes', and we also exploit the combined approach for $\mathcal{ELHO}^r_\bot$ (see Section 4.2) implemented in KARMA.[5]

PAGOdA accepts as input arbitrary OWL 2 DL ontologies, datasets in turtle format[6] and CQs in SPARQL. Queries can be interpreted under ground or certain answer semantics. In the former case, PAGOdA is sound and complete. In the latter case, however, PAGOdA is limited by the capabilities of HermiT, which can only check entailment of ground or DL concept queries; hence, PAGOdA can guarantee completeness only if the lower and upper bounds match, or if the query can be transformed into a DL concept query via internalisation (see Section 2.3). Otherwise, PAGOdA returns a sound (but possibly incomplete) set of answers, along with a bound on the incompleteness of the computed answer set.

The architecture of PAGOdA is depicted in Figure 4. Each box in Figure 4 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 CQ evaluation and the incremental addition of facts, and any fully-fledged OWL 2 DL reasoner that supports fact entailment.

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 4, the 'loading ontology' and 'materialisation' steps are query independent. Therefore, both of them are counted as *pre-processing* steps. 'Computing query bounds', 'extracting subset' and 'full 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., 2009). 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 4.1), 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 4 as ellipses). The lower bound component performs Steps 1 and 2 from Section 4.3 in order to compute an aggregated lower bound M^L_2. The c-chase and c-chase^f components compute the M^U_2 and M^U_3 upper bound materialisations as described in Section 5.4 using a dedicated implementation of the c-chase algorithm. The chase sequence

5. KARMA is available at `http://www.cs.ox.ac.uk/isg/tools/KARMA/`.
6. http://www.w3.org/TR/turtle/

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 5.4, 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 4.1). If a fact of the form $\overline{P}(\vec{a})$ is obtained in the materialisation, then $\neg P(\vec{a})$ follows from the knowledge base. Hence, if we have obtained $\overline{P}(\vec{a})$, then we try to avoid choosing $P(\vec{a})$ from a disjunct $P(\vec{a}) \vee \chi$ 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 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. Given a query q, PAGOdA uses the M_2^L lower bound materialisation to compute the lower bound answer L^q. In order to do this it exploits KARMA's implementation of the filtration procedure (algorithm soundAnswers in Section 4.2), but for clarity this step is shown separately (as a circle with an "F" in it) in Figure 4. If $\bot_s$ was not derived when computing the M_3^U materialisation, $U^q = \mathsf{cert}(q, M_2^U) \cap \mathsf{cert}(q, M_3^U)$; otherwise $U^q = \mathsf{cert}(q, M_2^U)$. In either case U^q is computed directly by using RDFox to answer q w.r.t. the relevant materialisation.

Extracting subsets. The tracking encoder component implements the datalog encoding based on Definition 6.1 with the optimisations described in Section 6.3. The resulting datalog knowledge base is added to the rules and data 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.

Full reasoning. PAGOdA uses HermiT to verify gap answers in $G^q = U^q \setminus L^q$. As HermiT only accepts queries given either as facts or DL concepts, we have implemented the standard rolling-up technique to transform internalisable CQs. 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 7). The remaining gap answers $G' \subseteq G^q$ are then passed to the endormorphism checker, which exploits a greedy algorithm to compute a (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 7). Verified answers from G' are combined with the lower bound L^q to give $\mathsf{cert}(q, \mathcal{O} \cup \mathcal{D})$.

9. Related Work

Conjunctive query answering over ontology-enriched datasets 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.

9.1 Computational Complexity of CQ Answering

The decision problem associated to CQ answering is *conjunctive query entailment* (CQE), namely to decide whether $\mathcal{K} \models q(\vec{a})$ when given as input a CQ q, a possible answer $\vec{a}$, 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 in (Rudolph & Glimm, 2010) under the assumption that the query does not mention transitive relations. Decidability of CQE for unrestricted OWL DL or OWL 2 DL ontologies and CQs 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, Lutz, Ortiz, & Simkus, 2009). Hardness results for 2-ExpTime can be obtained already for $\mathcal{ALCI}$ (Lutz, 2008) as well as for the logic Horn-$\mathcal{SROIQ}$ which underpins the Horn fragment of OWL 2 DL (Ortiz, Rudolph, & Simkus, 2011). CQE for $\mathcal{ALC}$ and $\mathcal{SHQ}$, which do not involve inverse roles, 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).

Given 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., 2009). CQE in the OWL 2 EL profile is PSpace-complete (Stefanoni, Motik, Krötzsch, & Rudolph, 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 & 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). 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, data complexity is known to be in AC^0 for the OWL 2 QL profile (Calvanese et al., 2007).

The complexity of CQE is also well understood for rule-based KR languages. For plain datalog, it is ExpTime-complete in combined complexity and PTime-complete w.r.t. data complexity. 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 KR languages based on existential rules (Calì, Gottlob, & Lukasiewicz, 2012). This includes guarded (Calì et al., 2013), sticky (Calì, Gottlob, & Pieris, 2011), and acyclic (Cuenca Grau et al., 2013) datalog$^\pm$. The extension of datalog$^\pm$ languages with disjunctive rules has been recently studied in (Alviano et al., 2012; Bourhis et al., 2013).

Finally, we refer to ground query entailment (GCQE) as the problem of checking whether a tuple $\vec{a}$ is a ground answer to $q(\vec{x}) = \exists \vec{y} \varphi(\vec{x}, \vec{y})$ w.r.t. $\mathcal{K}$. In KR languages that allow for existentially quantified 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 DL.

9.2 Practical Query Answering Approaches

Some off-the-shelf DL reasoners, such as Pellet (Sirin et al., 2007) and HermiT (Glimm et al., 2014) provide support for query answering. Pellet supports SPARQL conjunctive queries and also implements the rolling-up technique. In contrast, HermiT does not provide a SPARQL API and it only supports CQs in the form of (complex) DL concepts. Racer was among the first DL reasoners to implement and optimise CQ answering under ground semantics (Haarslev, Hidde, Möller, & Wessel, 2012). Finally, there has also been intensive work on optimising query answering in DL systems, including filter-and-refine techniques (Wandelt et al., 2010), ordering strategies of query atoms (Kollia & Glimm, 2013), and data summarisation (Dolby et al., 2009). Optimising CQ 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 queries that require full reasoning.

RDF triple stores typically implement materialisation-based (a.k.a. forward chaining) reasoning algorithms, and answer queries by evaluating them over the resulting materialisation. Jena (McBride, 2001) and Sesame (Broekstra, Kampman, & van Harmelen, 2002) were among the first such systems to provide support for RDF-Schema. Modern triple stores such as OWLim (Bishop et al., 2011), and Oracle's native inference engine (Wu et al., 2008), provide extended suppport for ontologies in the RL profile. Additionally, RDFox (Motik et al., 2014) supports arbitrary datalog over unary and binary predicates. Finally, ASP engines such as DLV (Leone, Pfeifer, Faber, Eiter, Gottlob, Perri, & Scarcello, 2006) implement sound and complete reasoning for (extensions of) disjunctive datalog. Although triple stores exhibit appealing scalability, they can support only restricted ontology languages; however, as with DL reasoners, improving the scalability of triple stores is complementary to our approach, and advances in this area can be directly exploited in PAGOdA.

CQ answering over Horn ontologies is often realised by means of *query rewriting* techniques. A rewriting of a query q w.r.t. an ontology $\mathcal{O}$ is another query q' that captures all information from $\mathcal{O}$ necessary to answer q over an arbitrary dataset. Unions of CQs and datalog are common target languages for query rewriting. Query rewriting enables the reuse of optimised data management system: UCQs can be answered using standard relational databases, whereas datalog queries can be evaluated using a triple store. Query rewriting has been successfully applied to OWL 2 QL ontologies, where rewritability into UCQs is guaranteed. Example systems include QuOnto (Acciarri, Calvanese, De Giacomo, Lembo, Lenzerini, Palmieri, & Rosati, 2005) and Mastro (Calvanese, De Giacomo, Lembo, Lenzerini, Poggi, Rodriguez-Muro, Rosati, Ruzzi, & Savo, 2011), Rapid (Chortaras, Trivela, &

Stamou, 2011), Prexto (Rosati, 2012), and Ontop (Bagosi, Calvanese, Hardi, Komla-Ebri, Lanti, Rezk, Rodriguez-Muro, Slusnys, & Xiao, 2014). Datalog-based query rewriting has been implemented in systems such as REQUIEM (Pérez-Urbina, Motik, & Horrocks, 2010). However, although some of these systems have been successful in large scale applications, they are only applicable to Horn ontology languages, and even for OWL 2 QL the size of the rewriting can be exponential in the size of the ontology (Calvanese et al., 2007).

A technique for CQ answering over lightweight DLs that is receiving increasing attention is the so-called *combined approach* (Lutz, Toman, & Wolter, 2009; Stefanoni et al., 2013; Kontchakov, Lutz, Toman, Wolter, & Zakharyaschev, 2011). In the combined approach the dataset is first augmented with new facts in a query-independent way to build (in polynomial time) a model of the ontology. This model can be exploited for query answering in two equivalent ways. In the approach by (Lutz et al., 2009; Kontchakov et al., 2011) the query is first rewritten and then evaluated against the constructed model. Alternatively, in (Stefanoni et al., 2013; Lutz et al., 2013) the query is first evaluated over the model and then unsound answers are eliminated by means of a polynomial time *filtration* process. Combined approaches have been applied to logics of the $\mathcal{EL}$ family (Lutz et al., 2009; Stefanoni et al., 2013) as well as DL-Lite (Kontchakov et al., 2011), and in PAGOdA, we use the implementation of (Stefanoni et al., 2013) to compute the aggregated lower bound. However, as in query rewriting techniques, the combined approach is only applicable to Horn ontology languages.

Finally, similarly to PAGOdA, the system Hydrowl (Stoilos, 2014a) combines an OWL 2 RL reasoner with a query rewriting system and a fully-fledged DL reasoner in order to answer conjunctive queries over an OWL 2 knowledge base. 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, 2014b) and query rewriting, and is applicable only to ontologies for which a suitable repair exists. The second strategy exploits a *query base*: a set of atomic queries that Hydrowl computes in a pre-processing phase, and that can be fully answered using the triple store for the given ontology and an arbitrary dataset. When answering a query q, Hydrowl checks if q is "covered" by query base (Stoilos & 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. However, the computation of the query base does not appear to be correct in general,[7] and we believe that this accounts for the apparent incompleteness of Hydrowl in some of our tests (see Section 10.3.1).

9.3 Approximate Reasoning

The idea of transforming the ontology, data and/or query to obtain lower and upper bound answers has been already explored in previous work. The Screech system (Tserendorj et al., 2008) uses KAON2 (Hustadt, Motik, & Sattler, 2007) to transform a $\mathcal{SHIQ}$ ontology into a (exponential size) disjunctive datalog program in such a way that ground answers to queries are preserved. Subsequently, Screech can exploit (unsound or incomplete) techniques to transform disjunctive datalog into plain datalog. In this way, Screech computes only an approximation of the answer. TrOWL (Thomas et al., 2010) exploits approximation techniques to transform an OWL 2 ontology into an ontology in the QL profile (Pan &

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

	♯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 3: Statistics for test datasets

Thomas, 2007). The approximation first computes the closure of the input ontology under entailment of OWL 2 QL axioms, and then disregards all axioms outside OWL 2 QL. Related approximations into OWL 2 QL have also been proposed, e.g., by Wandelt et al. (2010) and Console et al. (2014). Efficient approximation strategies for OWL 2 ontologies are again complementary to our approach, as they can be exploited by PAGOdA in order to refine lower and upper bound query answers.

10. Evaluation

We have evaluated our query answering system PAGOdA on a 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 Sections 10.1 and 10.2, respectively. Our results are discussed in Section 10.3. 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.[8]

10.1 Test Ontologies and Queries

Table 3 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 colummns of the table, respectively. Finally, the rightmost column lists the number of data facts in each dataset.

LUBM and UOBM are widely-used reasoning benchmarks (Guo, Pan, & Heflin, 2005; Ma, Yang, Qiu, Xie, Pan, & Liu, 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

8. `http://www.cs.ox.ac.uk/isg/tools/PAGOdA/2015/jair/`

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.[9] 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 & 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.

ChEMBL, Reactome, and Uniprot are realistic ontologies that have been made publicly available through the European Bioinformatics Institute (EBI) linked data platform.[10] 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 & Faloutsos, 2006) and computed subsets of the data of increasing size. We have used for evaluation those example queries that correspond to CQs as well as all atomic queries over the relevant signature.

10.2 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 DL ontologies, including ontology consistency checking and concept instance retrieval. Additionally, all but HermiT provide support for SPARQL queries.

As pointed out in the Section 9, 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

9. http://www.virtualflybrain.org/site/vfb_site/overview.htm
10. http://www.ebi.ac.uk/rdf/platform

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., 2009; Glimm et al., 2014). HermiT focuses on standard reasoning tasks in DLs. It does not provide a SPARQL or conjunctive query answering API, but it is capable of answering atomic queries over unary predicates and checking fact entailment.

Pellet is a tableau-based OWL 2 DL reasoner with support for CQ 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 expressed in SPARQL. Pellet is also capable of computing all certain answers to internalisable conjunctive queries using the rolling-up technique (see Section 2.3).

TrOWL is a system based on approximated reasoning. It accepts as input an arbitrary OWL 2 DL ontology and a CQ in SPARQL, and aims at computing all 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, 2014a) is a hybrid reasoning system that is similar in spirit to PAGOdA (see Section 9.2 for a detailed comparison). Hydrowl integrates the triple store OWLim and HermiT. It accepts as input an arbitrary OWL 2 ontology and conjunctive queries as rules, and then computes the ground answers to the query.

10.3 Experiments and Results

We have performed three different experiments. In the first experiment, 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 experiment, we evaluated the *scalability* by considering datasets of increasing size. Finally, in the third experiment, we evaluated the *effectiveness* of each of the different reasoning techniques implemented in PAGOdA.

10.3.1 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 internalisation (see Section 2.3) to additionally compute all certain answers. In the case of FLY, all test queries yield an empty set of ground answers, so in this case we computed only the certain answers (all FLY queries can be internalised). We set timeouts of 20 minutes for answering each individual query, and 5 hours for answering all the queries over a given ontology.

Figure 5 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

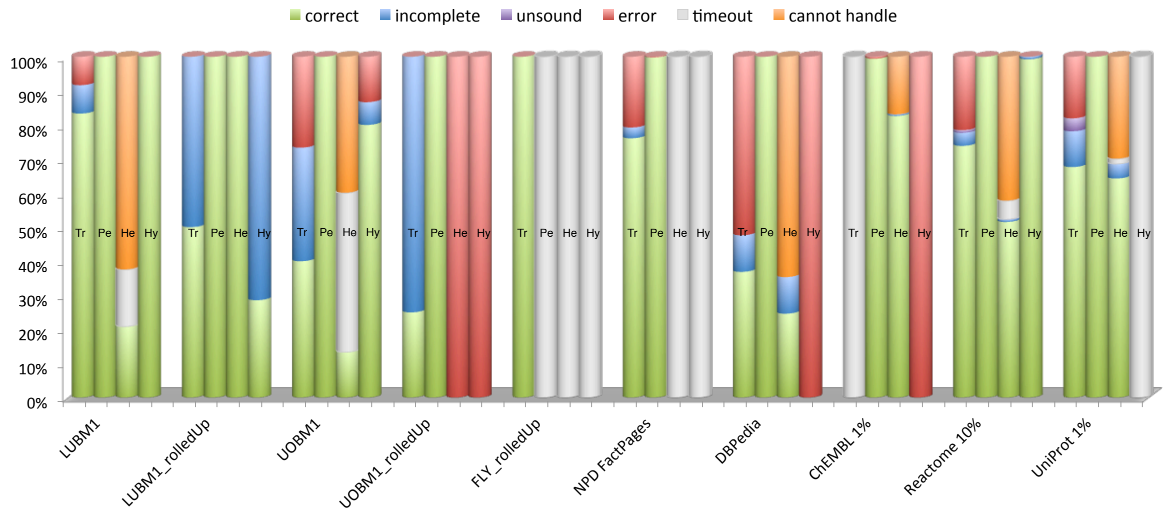


Figure 5: Quality of the answers computed by each system. The four bars for each ontology represent Trowl, Pellet, HermiT and Hydrowl respectively.

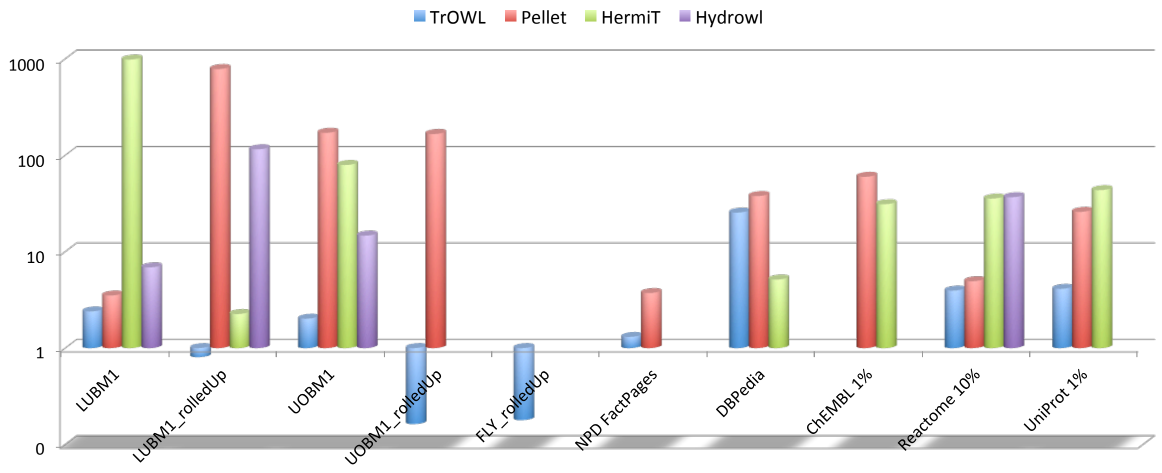


Figure 6: Performance comparison with other systems. Each bar depicts the total time to answer all test queries for the relevant ontology in comparison with PAGOdA.

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 timeout. 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 6 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 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 distinguished 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.

10.3.2 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 10.1 for each of these ontologies; as in Section 10.3.1, we computed ground answers and, whenever possible, used internalisation to additionally compute certain answers. For each test ontology we measured the following:

- *Pre-processing time.* This includes all pre-processing steps in Section 8 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.

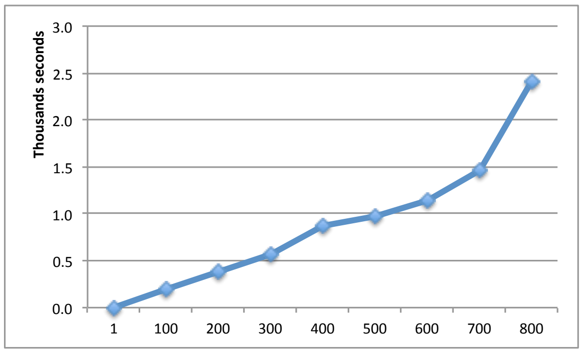
In the scalability test, we set a timeout of 5 hours for answering all queries and 2.5 hours for each individual query. For LUBM and UOBM, we increased the size of the dataset until PAGOdA exceeded the timeout; for the other ontologies, PAGOdA was able to answer all queries within the timeout, even with the largest dataset.

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.

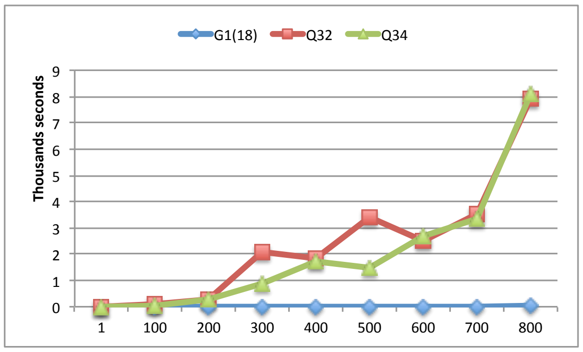
Our results are summarised in Figures 7 and 8. For each ontology, we plot time against the size of the input dataset, and for query processing we distinguish different groups of queries as discussed above. PAGOdA behaves relatively uniformly for queries in **G1** and **G2**, so we plot only the average 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 7a, 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 7b illustrates the average query processing time for queries in **G1**, which never exceeds 13 seconds, as 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.

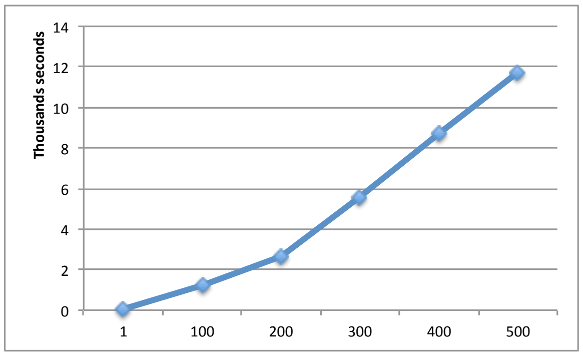
UOBM(n) As shown in Figure 7c, pre-processing times are significantly higher than for LUBM, reflecting the increased complexity of the ontology, but still appear to scale linearly with dataset size. As with LUBM, most test queries were contained in **G1**, and their processing times never exceeds 8 seconds 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 7d 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).



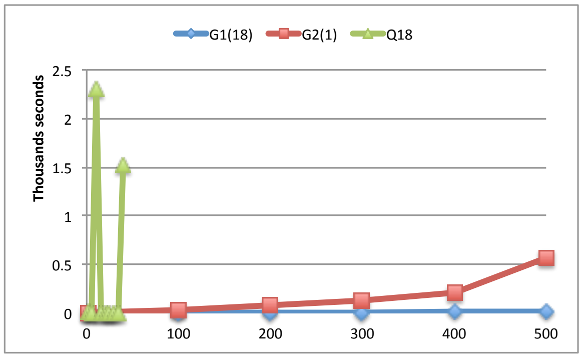
(a) LUBM pre-processing



(b) LUBM query processing



(c) UOBM pre-processing



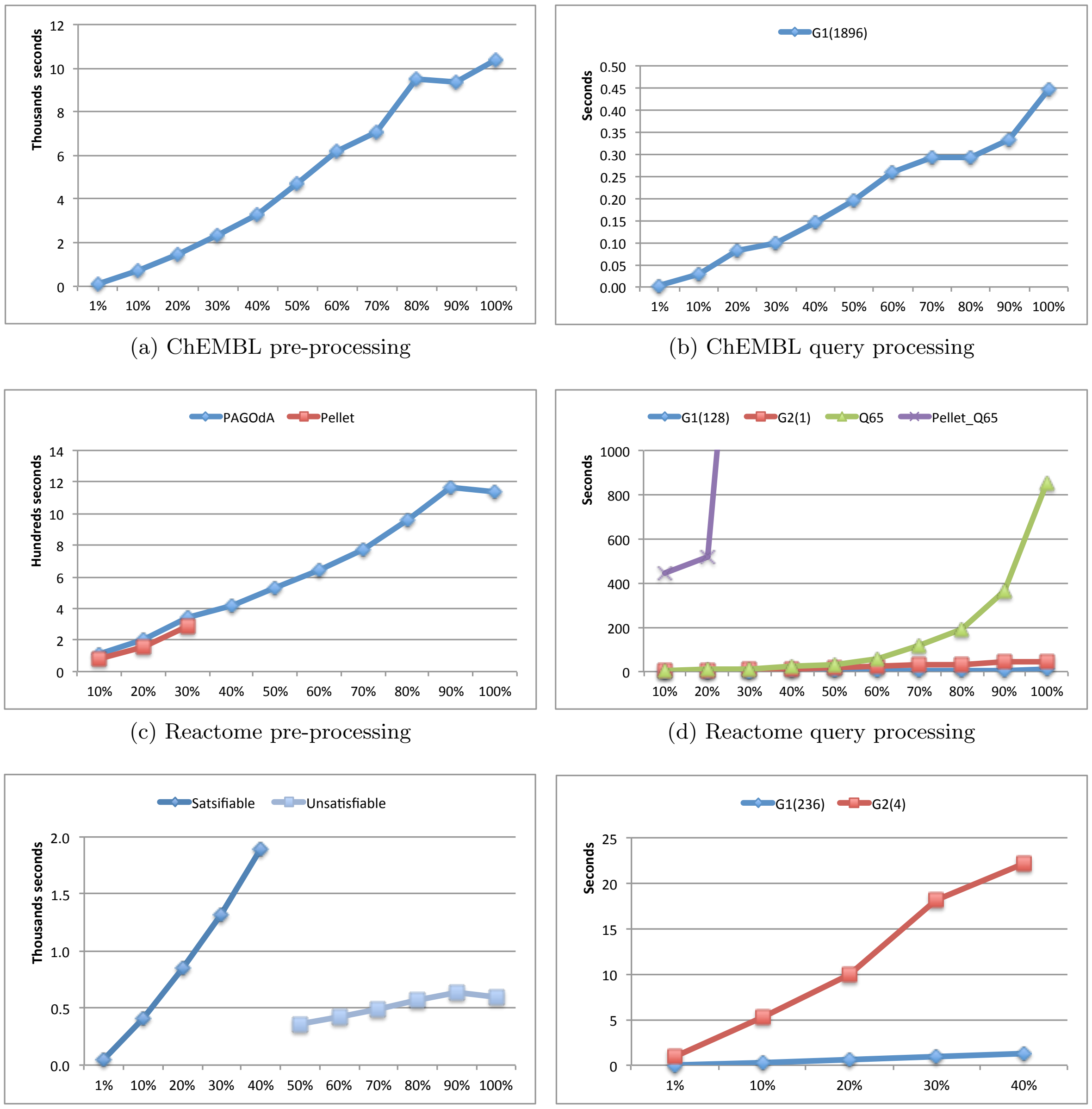
(d) UOBM query processing

Figure 7: Scalability tests on benchmarks

ChEMBL As shown in Figure 8a, pre-processing times are significant but manageable, and again appear to scale linearly with dataset size. All test queries were contained in **G1**. Figure 8b 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 8c, 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 8d. 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 8c. Average query-processing times for queries in **G1** and **G2** are slightly higher than those of PAGOdA. In contrast, times for query Q65 were significantly higher: 445s, 518s and $2,626$s for Reactome $10\%, 20\%$ and 30%, respectively (see Figure 8d). Processing times for Q65 in PAGOdA,



(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 8: Scalability tests on EBI linked data platform

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 8e 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

	LUBM (100)	UOBM (1)	FLY	NPD	DBPedia	ChEMBL 1%	Reactome 10%	Uniprot 1%
Total	35	20	6	478	1247	1896	130	240
$L_1 + U_1$	26	4	0	442	1240	1883	82	204
$L_2 + U_1$	33	4	5	442	1241	1883	82	204
$L_2 + U_2$	33	12	5	442	1241	1883	98	204
$L_2 + U_{2\|3}$	33	16	5	473	1246	1896	128	236

Table 4: ♯Queries answered by different bounds

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 8f). 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.

10.3.3 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 Sections 4 and 5 we described different techniques for computing lower and upper bound query answers. Table 4 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 4.1 as L_1 and to the aggregated lower bound described in Section 4.3 as L_2. Similarly, we refer to the three upper bound computation techniques discussed in Section 5.4 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 5.2, 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 5.3, 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.

	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 5: Size of the largest subsets given as percentage over input rules and facts.

	LUBM		UOBM				FLY	DBPedia	NPD	Reactome		UniProt
$L_2 + U_{2\vert 3}$	26	14	264	112	1470	264	344	10	326	18	52	168
+ Sum	26	14	264	0	1444	264	344	0	0	0	52	0
+ Dep	1	1	1	0	1	1	7	0	0	0	37	0

Table 6: The number of hard calls to HermiT to fully answer each query

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 5 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 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 6 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.

11. Conclusions

In this paper, we have investigated a novel 'pay-as-you-go' approach to conjunctive query answering that combines a datalog reasoner with a fully-fledged reasoner. The key feature of our approach is that it delegates the bulk of the computation to the datalog reasoner and resorts to the fully-fledged reasoner only as necessary to fully answer the query.

The reasoning techniques we have proposed here are very general and are applicable to a wide range of knowledge representation languages. Our main goal in practice, however, has been to realise our approach in a highly scalable and robust query answering system for OWL 2 DL ontologies, which we have called PAGOdA. Our extensive evaluation has not only confirmed the feasibility of our approach in practice, but also that our system PAGOdA significantly ourperforms state-of-the art reasoning systems in terms of both robustness and scalability. In particular, our 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.

Acknowledgments

This is an extended version of the papers 'Pay-As-You-Go OWL Query Answering Using a Triple Store' published at AAAI 2014, and 'Complete Query Answering over Horn Ontologies Using a Triple Store' published at ISWC 2013.

This work has been supported by the Royal Society under a Royal Society Research Fellowship, by the EPSRC projects MaSI^3, Score! and DBOnto, and by the EU FP7 project Optique.

References

Abiteboul, S., Hull, R., & Vianu, V. (Eds.). (1995). *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Acciarri, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Palmieri, M., & Rosati, R. (2005). QuOnto: Querying ontologies. In Veloso, M. M., & Kambhampati, S. (Eds.), *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*, pp. 1670–1671. AAAI Press / The MIT Press.

Alviano, M., Faber, W., Leone, N., & Manna, M. (2012). Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues. *Theory and Practice of*

Logic Programming, *12*(4-5), 701–718.

Baader, F., Brandt, S., & Lutz, C. (2005). Pushing the $\mathcal{EL}$ envelope. In *IJCAI 2015, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pp. 364–369.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge Univ. Press.

Bagosi, T., Calvanese, D., Hardi, J., Komla-Ebri, S., Lanti, D., Rezk, M., Rodriguez-Muro, M., Slusnys, M., & Xiao, G. (2014). The Ontop framework for ontology based data access. In Zhao, D., Du, J., Wang, H., Wang, P., Ji, D., & Pan, J. Z. (Eds.), *CSWS 2014, Proceedings of the Semantic Web and Web Science - 8th Chinese Conference, Wuhan, China, August 8-12, 2014, Revised Selected Papers*, Vol. 480 of *Communications in Computer and Information Science*, pp. 67–77. Springer.

Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., & Velkov, R. (2011). OWLIM: A family of scalable semantic repositories. *Semantic Web*, *2*(1), 33–42.

Bourhis, P., Morak, M., & Pieris, A. (2013). 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*, pp. 796–802. AAAI Press.

Broekstra, J., Kampman, A., & van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying RDF and RDF schema. In Horrocks, I., & Hendler, J. A. (Eds.), *ISWC 2002, Proceedings the Semantic Web - First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, Vol. 2342 of *Lecture Notes in Computer Science*, pp. 54–68. Springer.

Calì, A., Gottlob, G., & Kifer, M. (2013). Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research*, *48*, 115–174.

Calì, A., Gottlob, G., & Lukasiewicz, T. (2012). A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, *14*, 57–83.

Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., & Pieris, A. (2010). Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS 2010, Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, 11-14 July 2010, Edinburgh, United Kingdom*, pp. 228–242. IEEE Computer Society.

Calì, A., Gottlob, G., & Pieris, A. (2011). New expressive languages for ontological query answering. In Burgard, W., & Roth, D. (Eds.), *AAAI 2011, Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, San Francisco, California, USA, August 7-11, 2011*, Vol. 2, pp. 1541–1546. AAAI Press.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., & Savo, D. F. (2011). The MASTRO system for ontology-based data access. *Semantic Web*, *2*(1), 43–53.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2007). Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *Journal of Automated Reasoning*, *39*(3), 385–429.

Chortaras, A., Trivela, D., & Stamou, G. B. (2011). Optimized query rewriting for OWL 2 QL. In Bjørner, N., & Sofronie-Stokkermans, V. (Eds.), *CADE 23, Proceedings of the 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011*, Vol. 6803 of *Lecture Notes in Computer Science*, pp. 192–206. Springer.

Console, M., Mora, J., Rosati, R., Santarelli, V., & Savo, D. F. (2014). 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*, pp. 164–179.

Cuenca Grau, B., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., & Wang, Z. (2013). Acyclicity notions for existential rules and their application to query answering in ontologies. *Journal of Artificial Intelligence Research*, *47*, 741–808.

Cuenca Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P. F., & Sattler, U. (2008). OWL 2: The next step for OWL. *Journal of Web Semantics*, *6*(4), 309–322.

Cuenca Grau, B., Motik, B., Stoilos, G., & Horrocks, I. (2012). Completeness guarantees for incomplete ontology reasoners: Theory and practice. *Journal of Artificial Intelligence Research*, *43*, 419–476.

Dantsin, E., Eiter, T., Gottlob, G., & Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Computing Surveys*, *33*(3), 374–425.

Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., & Ma, L. (2007). 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*, pp. 299–304. AAAI Press.

Dolby, J., Fokoue, A., Kalyanpur, A., Schonberg, E., & Srinivas, K. (2009). Scalable highly expressive reasoner (SHER). *Journal of Web Semantics*, *7*(4), 357–361.

Eiter, T., Fink, M., Tompits, H., & Woltran, S. (2004). 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*, pp. 87–99.

Eiter, T., Lutz, C., Ortiz, M., & Simkus, M. (2009). Query answering in description logics with transitive roles. In Boutilier, C. (Ed.), *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 759–764.

Eiter, T., Ortiz, M., & Simkus, M. (2012). Conjunctive query answering in the description logic $\mathcal{SH}$ using knots. *Journal of Computer and System Sciences*, *78*(1), 47–85.

Erling, O., & Mikhailov, I. (2009). Virtuoso: RDF support in a native RDBMS. In Virgilio, R. D., Giunchiglia, F., & Tanca, L. (Eds.), *Semantic Web Information Management - A Model-Based Perspective*, pp. 501–519. Springer.

Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HermiT: An OWL 2 reasoner. *Journal of Automated Reasoning*, *53*(3), 245–269.

Glimm, B., Lutz, C., Horrocks, I., & Sattler, U. (2008). Conjunctive query answering for the description logic $\mathcal{SHIQ}$. *Journal of Artificial Intelligence Research*, *31*, 157–204.

Grosof, B. N., Horrocks, I., Volz, R., & Decker, S. (2003). Description logic programs: combining logic programs with description logic. In Hencsey, G., White, B., Chen, Y. R., Kovács, L., & Lawrence, S. (Eds.), *WWW 2003, Proceedings of the Twelfth International World Wide Web Conference, Budapest, Hungary, May 20-24, 2003*, pp. 48–57. ACM.

Guo, Y., Pan, Z., & Heflin, J. (2005). LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, *3*(2-3), 158–182.

Haarslev, V., Hidde, K., Möller, R., & Wessel, M. (2012). The RacerPro knowledge representation and reasoning system. *Semantic Web*, *3*(3), 267–277.

Horrocks, I., Kutz, O., & Sattler, U. (2006). 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*, pp. 57–67.

Horrocks, I., Patel-Schneider, P. F., & van Harmelen, F. (2003). From $\mathcal{SHIQ}$ and RDF to OWL: the making of a web ontology language. *Journal of Web Semantics*, *1*(1), 7–26.

Horrocks, I., & Tessaris, S. (2000). A conjunctive query language for description logic aboxes. In Kautz, H. A., & Porter, B. W. (Eds.), *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.*, pp. 399–404. AAAI Press / The MIT Press.

Hustadt, U., Motik, B., & Sattler, U. (2007). Reasoning in description logics by a reduction to disjunctive datalog. *Journal of Automated Reasoning*, *39*(3), 351–384.

Jiménez-Ruiz, E., & Cuenca Grau, B. (2011). LogMap: Logic-based and scalable ontology matching. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N. F., & Blomqvist, E. (Eds.), *ISWC 2011, The Semantic Web - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, Vol. 7031 of *Lecture Notes in Computer Science*, pp. 273–288. Springer.

Kaminski, M., Nenov, Y., & Grau, B. C. (2014). Computing datalog rewritings for disjunctive datalog programs and description logic ontologies. In Kontchakov, R., & Mugnier, M. (Eds.), *Web Reasoning and Rule Systems - 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings*, Vol. 8741 of *Lecture Notes in Computer Science*, pp. 76–91. Springer.

Kollia, I., & Glimm, B. (2013). Optimizing SPARQL query answering over OWL ontologies. *Journal of Artificial Intelligence Research*, *48*, 253–303.

Kontchakov, R., Lutz, C., Toman, D., Wolter, F., & Zakharyaschev, M. (2011). The combined approach to ontology-based data access. In Walsh, T. (Ed.), *IJCAI 2011,* *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 2656–2661. IJCAI/AAAI.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., & Scarcello, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, *7*(3), 499–562.

Leskovec, J., & Faloutsos, C. (2006). 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*, pp. 631–636.

Lutz, C. (2008). The complexity of conjunctive query answering in expressive description logics. In Armando, A., Baumgartner, P., & Dowek, G. (Eds.), *IJCAR 2008, Proceedings of the 4th International Joint Conference Automated Reasoning, Sydney, Australia, August 12-15, 2008*, Vol. 5195 of *Lecture Notes in Computer Science*, pp. 179–193. Springer.

Lutz, C., Seylan, I., Toman, D., & Wolter, F. (2013). The combined approach to OBDA: Taming role hierarchies using filters. In Alani, H., Kagal, L., Fokoue, A., Groth, P. T., Biemann, C., Parreira, J. X., Aroyo, L., Noy, N. F., Welty, C., & Janowicz, K. (Eds.), *ISWC 2013, Proceedings of the Semantic Web - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, Vol. 8218 of *Lecture Notes in Computer Science*, pp. 314–330. Springer.

Lutz, C., Toman, D., & Wolter, F. (2009). Conjunctive query answering in the description logic EL using a relational database system. In Boutilier, C. (Ed.), *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 2070–2075.

Ma, L., Yang, Y., Qiu, Z., Xie, G. T., Pan, Y., & Liu, S. (2006). Towards a complete OWL ontology benchmark. In Sure, Y., & Domingue, J. (Eds.), *ESWC 2006, The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, Budva, Montenegro, June 11-14, 2006, Proceedings*, Vol. 4011 of *Lecture Notes in Computer Science*, pp. 125–139. Springer.

Manola, F., & Miller, E. (2004). RDF primer. W3C Recommendation. Available at http://www.w3.org/TR/rdf-primer/.

Marnette, B. (2009). 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*, pp. 13–22.

McBride, B. (2001). Jena: Implementing the RDF model and syntax specification. In *SemWeb 2001, Proceedings of the Second International Workshop on the Semantic Web*.

Möller, R., Neuenstadt, C., Özcep, Ö. L., & Wandelt, S. (2013). Advances in accessing big data with expressive ontologies. In Timm, I. J., & Thimm, M. (Eds.), *KI 2013, Proceedings of Advances in Artificial Intelligence - 36th Annual German Conference on AI, Koblenz, Germany, September 16-20, 2013*, Vol. 8077 of *Lecture Notes in Computer Science*, pp. 118–129. Springer.

Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., & Lutz, C. (2009). OWL 2 Web Ontology Language Profiles. W3C Recommendation. Available at http://www.w3.org/TR/owl2-profiles/.

Motik, B., Nenov, Y., Piro, R., Horrocks, I., & Olteanu, D. (2014). Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In Brodley, C. E., & Stone, P. (Eds.), *AAAI 2014, Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pp. 129–137. AAAI Press.

Motik, B., Shearer, R., & Horrocks, I. (2009). Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, *36*, 165–228.

Ortiz, M., Rudolph, S., & Simkus, M. (2011). 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*, pp. 1039–1044.

Pan, J. Z., & Thomas, E. (2007). 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*, pp. 1434–1439.

Pérez-Urbina, H., Motik, B., & Horrocks, I. (2010). Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, *8*(2), 186–209.

Robinson, J. A., & Voronkov, A. (Eds.). (2001). *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press.

Rodriguez-Muro, M., & Calvanese, D. (2012). High performance query answering over *DL-Lite* ontologies. In Brewka, G., Eiter, T., & McIlraith, S. A. (Eds.), *KR 2012, Proceedings of Principles of Knowledge Representation and Reasoning, the Thirteenth International Conference, Rome, Italy, June 10-14, 2012*, pp. 308–318. AAAI Press.

Rosati, R. (2012). Prexto: Query rewriting under extensional constraints in DL - lite. In Simperl, E., Cimiano, P., Polleres, A., Corcho, Ó., & Presutti, V. (Eds.), *ESWC 2012, Proceedings of the Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, Heraklion, Crete, Greece, May 27-31, 2012*, Vol. 7295 of *Lecture Notes in Computer Science*, pp. 360–374. Springer.

Rudolph, S., & Glimm, B. (2010). Nominals, inverses, counting, and conjunctive queries or: Why infinity is your friend!. *Journal of Artificial Intelligence Research*, *39*, 429–481.

Schaerf, A. (1993). On the complexity of the instance checking problem in concept languages with existential quantification. In Komorowski, H. J., & Ras, Z. W. (Eds.), *ISMIS 1993, Proceedings of Methodologies for Intelligent Systems, 7th International Symposium, Trondheim, Norway, June 15-18, 1993*, Vol. 689 of *Lecture Notes in Computer Science*, pp. 508–517. Springer.

Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, *5*(2), 51–53.

Staab, S., & Studer, R. (Eds.). (2004). *Handbook on Ontologies*. International Handbooks on Information Systems. Springer.

Stefanoni, G., & Motik, B. (2015). Answering conjunctive queries over EL knowledge bases with transitive and reflexive roles. In Bonet, B., & Koenig, S. (Eds.), *AAAI 2015, Proceedings of the 29th AAAI Conference on Artificial Intelligence*, Austin, TX, USA. AAAI Press. To appear.

Stefanoni, G., Motik, B., & Horrocks, I. (2013). Introducing nominals to the combined query answering approaches for $\mathcal{EL}$. In *AAAI 2013, Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pp. 1177–1183.

Stefanoni, G., Motik, B., Krötzsch, M., & Rudolph, S. (2014). The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. *Journal of Artificial Intelligence Research*, *51*, 645–705.

Stoilos, G. (2014a). 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*, pp. 230–238.

Stoilos, G. (2014b). Ontology-based data access using rewriting, OWL 2 RL systems and repairing. In Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., & Tordai, A. (Eds.), *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, Vol. 8465 of *Lecture Notes in Computer Science*, pp. 317–332. Springer.

Stoilos, G., & Stamou, G. B. (2014). Hybrid query answering over OWL ontologies. In Schaub, T., Friedrich, G., & O'Sullivan, B. (Eds.), *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, Vol. 263 of *Frontiers in Artificial Intelligence and Applications*, pp. 855–860. IOS Press.

Thomas, E., Pan, J. Z., & Ren, Y. (2010). 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*, pp. 431–435.

Tserendorj, T., Rudolph, S., Krötzsch, M., & Hitzler, P. (2008). Approximate OWL-reasoning with screech. In Calvanese, D., & Lausen, G. (Eds.), *RR 2008, Proceedings of Web Reasoning and Rule Systems, Second International Conference, Karlsruhe, Germany, October 31-November 1, 2008*, Vol. 5341 of *Lecture Notes in Computer Science*, pp. 165–180. Springer.

W3C SPARQL Working Group (2013). SPARQL 1.1 Overview. W3C Recommendation. Available at http://www.w3.org/TR/sparql11-overview/.

Wandelt, S., Möller, R., & Wessel, M. (2010). Towards scalable instance retrieval over ontologies. *International Journal of Software and Informatics*, *4*(3), 201–218.

Wu, Z., Eadon, G., Das, S., Chong, E. I., Kolovski, V., Annamalai, M., & Srinivasan, J. (2008). Implementing an inference engine for RDFS/OWL constructs and user-defined rules in oracle. In Alonso, G., Blakeley, J. A., & Chen, A. L. P. (Eds.), *ICDE 2008, Proceedings of the 24th International Conference on Data Engineering, April 7-12, 2008, Cancún, México*, pp. 1239–1248. IEEE.