Making the Most of your Triple Store: Query Answering in OWL 2 Using an RL Reasoner

Yujiao Zhou
Computer Science Dept.
University of Oxford
yzhou@cs.ox.ac.uk

Bernardo Cuenca Grau
Computer Science Dept.
University of Oxford
berg@cs.ox.ac.uk

Ian Horrocks
Computer Science Dept.
University of Oxford
ian.horrocks@cs.ox.ac.uk

Zhe Wu
Oracle Corporation
alan.wu@oracle.com

Jay Banerjee
Oracle Corporation
jayanta.banerjee@oracle.com

ABSTRACT

Triple stores implementing the RL profile of OWL 2 are becoming increasingly popular. In contrast to unrestricted OWL 2, the RL profile is known to enjoy favourable computational properties for query answering, and state-of-the-art RL reasoners such as OWLim and Oracle's native inference engine of Oracle Spatial and Graph have proved extremely successful in industry-scale applications. The expressive restrictions imposed by OWL 2 RL may, however, be problematical for some applications. In this paper, we propose novel techniques that allow us (in many cases) to compute exact query answers using an off-the-shelf RL reasoner, even when the ontology is outside the RL profile. Furthermore, in the cases where exact query answers cannot be computed, we can still compute both lower and upper bounds on the exact answers. These bounds allow us to estimate the degree of incompleteness of the RL reasoner on the given query, and to optimise the computation of exact answers using a fully-fledged OWL 2 reasoner. A preliminary evaluation using the RDF Semantic Graph feature in Oracle Database has shown very promising results with respect to both scalability and tightness of the bounds.

1. INTRODUCTION

The success of RDF as a language for representing semi-structured data on the (semantic) Web has led to the proliferation of applications based on large repositories of data stored in RDF format. In these applications, access to data relies on queries formulated in the standard query language SPARQL [29]; additionally, background knowledge required to unambiguously specify the meaning of the data in the context of the application may be captured using the standard ontology language OWL 2 [23].

Efficient management and querying of such large data repositories is a core problem in the development of RDF-based applications. Significant progress has been made in recent years in the design and development of efficient RDF data management systems, and state-of-the-art systems such as Hexastore [34] and RDF-3X [26] have combined highly optimised data structures and query answering algorithms in order to achieve impressive performance. There have also been significant advances in clustering and data partitioning techniques [30, 13, 16], which allow RDF query engines to exploit various forms of parallel architecture. As a result, state-of-the-art RDF management systems are capable of dealing with very large data sets.

When an ontology is used to augment the semantics of the RDF data, query answers need to consider additional triples whose existence is *entailed* by the combination of the ontology and the data. *Materialisation-based* approaches are widely used to extend RDF data management systems to deal with this situation; they work by using forward chaining rules to materialise the entailed triples, and then evaluating queries over the resulting extended data set.

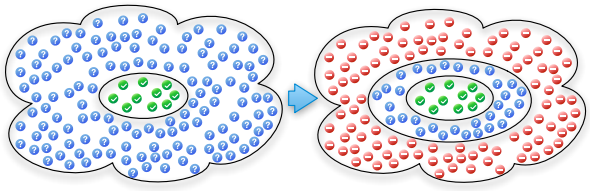
The success in practice of materialisation-based systems led to the development of the RL profile of OWL 2 [22], a large subset of OWL 2 for which query answering is known to be both theoretically tractable (in polynomial time w.r.t the size of the data), and practically realisable via materialisation. This combination of features has made OWL 2 RL increasingly popular, and state-of-the-art RL reasoners such as OWLim [1] and Oracle's RDF Semantic Graph [35] provide robust and scalable support for SPARQL query answering over OWL 2 RL ontologies and RDF data sets.

Although OWL 2 RL captures a substantial fragment of OWL 2, it necessarily restricts expressiveness. OWL 2 RL cannot, for example, capture *disjunctive knowledge* such as that expressed in the following axiom, which states that every student is either an undergraduate or a graduate student:

$$SubClassOf(\mathsf{Student}\ ObjectUnionOf(\mathsf{Grad}\ \mathsf{UnderGrad}));$$

nor can it capture *existentially quantified knowledge* such as that expressed in the following axiom, which states that each research assistant works for some research group:

$$SubClassOf(\mathsf{RA}\ SomeValuesFrom(\mathsf{works}\ \mathsf{Group})).$$



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

Figure 1: Combination of lower and upper bounds

These restrictions limit the applicability of OWL 2 RL in practice since disjunctive and existentially quantified statements abound in OWL ontologies. For example, the NCI Thesaurus contains many disjunctive statements, while ontologies such as SNOMED, FMA, and Fly Anatomy[1] contain thousands of existentially quantified statements.

Although the capabilities of RL reasoners are intrinsically limited, they are flexible enough to process ontologies outside OWL 2 RL on a 'best efforts' basis, as the materialisation rules effectively ignore those (parts of) axioms that are outside the RL profile. In such cases, answers to SPARQL queries are still guaranteed to be sound (the computed answer set includes only valid answer tuples), but may not be complete (the computed answer set may not include all valid answer tuples); thus, the answer set returned by the system can be thought of as a *lower bound* on the exact answers.

To ensure the completeness of query answers in such cases, one could abandon RL reasoners altogether in favour of fully fledged OWL 2 reasoners, such as HermiT [25], Pellet [32] and Racer [11]. However, despite intensive efforts at optimisation, the scalability of such systems falls far short of that exhibited by RL reasoners [20, 12].

In this paper, we propose novel techniques that allow us (in many cases) to compute sound and complete answers using off-the-shelf RL reasoners, even when the ontology is outside OWL 2 RL. Furthermore, in the cases where exact answers cannot be computed, our techniques allow us to compute an *upper bound* on the exact answers. This upper bound is useful in practice for (at least) two reasons. First, as illustrated in Figure 1(b), it allows us to bound the incompleteness of the RL reasoner by partitioning tuples into three sets: those that are definitely in the answer (marked with '✓'), those that may be in the answer (marked with '?'), and those that are definitely not in the answer (marked with '-'); without the upper bound no tuples can be ruled out, and the status of a potentially huge number of tuples is thus left undetermined (as illustrated in Figure 1(a)). Second, it allows us to optimise the computation of exact answers by checking—e.g., using a fully-fledged OWL 2 reasoner—only the (typically small number of) tuples remaining in the gap between the lower and upper bounds.

Our work is closely related to existing techniques for *theory approximation* [8, 31], where lower and upper bounds to query answers are obtained by transforming the knowledge base (and possibly also the query) into a less expressive language. Systems such as those described in [33, 28, 19], all of which we discuss in detail in Section 6, are able to compute upper bounds to query answers under certain conditions. To the best of our knowledge, however, our approach is the only one that enjoys all of the following desirable properties:

- In contrast to [33] and [28], computation of the upper bound requires only the ontology to be transformed, and is independent of both data and query.
- In contrast to [33], [28], and [19], the transformation increases the size of the ontology only linearly, and can be computed in linear time.
- In contrast to [28] and [19], which approximate the ontology into DL-Lite (i.e., OWL 2 QL), our approach uses OWL 2 RL, which will typically lead to tighter bounds, and allows us to directly exploit an industrial-strength OWL 2 RL reasoner as a "black box".
- In contrast to [33] and [19], our approach is independent of the query language, and hence can be applied not only to SPARQL queries, but also to more general languages such as (unions of) conjunctive queries.

An evaluation of our approach has been performed using Oracle's RDF Semantic Graph, a range of test data including both benchmark and realistic ontologies, and a variety of synthetic and realistic queries. The evaluation suggests that the gap between the lower and upper bounds is typically small, indeed often empty, and that the upper bound is usually tight (i.e., it coincides with the exact answers). Moreover, although computing the upper bound increased the cost of materialising the data set, it is still feasible for large scale data sets, and much more efficient than the computation of exact answers using an OWL 2 reasoner; indeed we believe that this is the first time that exact answers have been computed over data sets of this size and w.r.t. an ontology outside any of the OWL 2 profiles.

2. PRELIMINARIES

We adopt standard notions from first-order logic (FOL) with equality, such as variables, constants, terms, atoms, formulas, sentences, substitutions, satisfiability, unsatisfiability, and entailment (written $\models$). We use the standard notation $t \approx t'$ (an equality atom) to denote equality between terms and the standard abbreviation $t \not\approx t'$ for $\neg t \approx t'$ (an inequality atom). The *falsum* atom, which is evaluated to false in all interpretations, is denoted here as $\bot$, whereas the dual *truth* atom is represented as $\top$.

2.1 Ontologies and Data Sets

We assume basic familiarity with the OWL 2 and OWL 2 RL ontology languages [23, 22], as well as with the syntax and semantics of $\mathcal{SROIQ}$—the description logic (DL) underpinning OWL 2 (see [14] for details).[2]

In this paper, we exploit the normal form for $\mathcal{SROIQ}$ given in Definition 1. Each $\mathcal{SROIQ}$ TBox can be transformed into

[1] `http://obofoundry.org/cgi-bin/detail.cgi?id=fly_anatomy_xp`

[2] In this paper, we disregard datatypes for simplicity.

this normal form by introducing fresh predicates as needed (see [25] for details on the normalisation algorithm).

DEFINITION 1. *A $\mathcal{SROIQ}$-TBox is normalised if it contains only the following kinds of axioms, where $R_{(i)}$ are either an atomic role or the inverse of an atomic role:*

- *Concept inclusion axioms $\top \sqsubseteq \bigsqcup_{i=1}^{n} C_i$, where each C_i is of the form B, $\{c\}$, $\forall R.B$, $\exists R.\mathsf{Self}$, $\neg\exists R.\mathsf{Self}$, $\geq n\, R.B$, or $\leq n\, R.B$, with B either an atomic concept or the negation of an atomic concept, c an individual, and n a nonnegative integer;*
- *Role axioms $R_1 \sqsubseteq R_2$, $R_1 \circ R_2 \sqsubseteq R_3$, or $R_1 \sqcap R_2 \sqsubseteq \bot$.*

We deviate slightly from the treatment of ontologies given in the W3C specification of OWL 2, where there is no explicit distinction between schema (i.e., TBox) and data (i.e., ABox). It is often convenient, however, to think of the ontology as a TBox (i.e., as containing *only* schema axioms), and to treat the (RDF) data assertions separately; this makes no difference from a semantic point of view. We will, therefore, treat an OWL 2 ontology $\mathcal{O}$ as a $\mathcal{SROIQ}$-TBox, and assume that all assertions are in a separate data set D. W.l.o.g. we restrict ourselves in this paper to data sets consisting only of atoms, including inequalities but excluding $\bot$ and $\top$.

2.2 Queries

A *conjunctive query* (CQ), or simply a *query*, is a first-order formula of the form $Q(\vec{x}) = \exists \vec{y}.\varphi(\vec{x}, \vec{y})$, where Q is a distinguished query predicate and $\varphi(\vec{x}, \vec{y})$ is a conjunction of atoms different from $\bot$ and from an inequality. A tuple of constants $\vec{a}$ is an *answer* to $Q(\vec{x})$ w.r.t. a set $\mathcal{F}$ of first-order sentences and a set of ground atoms D if $\mathcal{F} \cup D \models Q(\vec{a})$. The answer set of $Q(\vec{x})$ w.r.t. $\mathcal{F}$ and D, which we often call the *exact answers* to the query, is denoted as $\mathsf{cert}(Q, \mathcal{F}, D)$, where the free variables of $Q(\vec{x})$ are omitted. SPARQL queries are semantically equivalent to a restricted class of CQs with no existential quantifiers.

2.3 Datalog Languages

The design of OWL 2 RL was inspired by Description Logic Programs [9] — a KR formalism that can be captured using either datalog [7] or DLs. Therefore, there exists a tight connection between datalog rules and OWL 2 RL axioms.

The main difference between OWL 2 and its RL profile is the ability to represent disjunctive and existentially quantified knowledge. Hence, there is a tight connection between OWL 2 and an extension of datalog, which we call datalog$^{\pm,\vee}$, where both existential quantifiers and disjunctions are allowed in the head of rules. The connection between OWL 2 and datalog$^{\pm,\vee}$ is relevant to us, since our approach uses datalog$^{\pm,\vee}$ rules as an intermediate representation of ontology axioms. We next define datalog$^{\pm,\vee}$ and postpone the description of its relationship with OWL 2 until Section 3.1.

A datalog$^{\pm,\vee}$ *rule* r is a first-order sentence of form (1)

$$\forall \vec{x}.[B_1 \wedge \ldots \wedge B_n] \rightarrow \bigvee_{i=1}^{m} \exists \vec{y}_i.\varphi_i(\vec{x}, \vec{y}_i) \qquad (1)$$

$$\begin{array}{ccccccc} \mathcal{O} & \rightsquigarrow & \Sigma_{\mathcal{O}} & \rightsquigarrow & \Xi(\Sigma_{\mathcal{O}}) & \rightsquigarrow & \mathcal{O}' \cup D'_{\mathcal{O}} \\ \vdots & & \vdots & & \vdots & & \vdots \\ \text{OWL 2} & & \text{DATALOG}^{\pm,\vee} & & \text{DATALOG} & & \text{OWL 2 RL} \end{array}$$

Figure 2: Transformation steps

where each B_j is an atom that is neither $\bot$ nor an inequality atom and whose free variables are contained in $\vec{x}$, and either

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

The quantifier $\forall \vec{x}$ is left implicit. The *body* of r is the set of atoms $\mathsf{body}(r) = \{B_1, \ldots, B_n\}$, and the *head* of r is the formula $\mathsf{head}(r) = \bigvee_{i=1}^{m} \exists \vec{y}_i.\varphi(\vec{x}, \vec{y}_i)$. A datalog$^{\pm,\vee}$ rule r is a datalog$^{\pm}$ rule if $m = 1$ [2], and it is a datalog rule if it is a datalog$^{\pm}$ rule and the head does not contain existentially quantified variables.[3]

For Σ a set of datalog rules and D a set of ground atoms, the *saturation* of Σ w.r.t. D is the set D' of all ground atoms entailed by $\Sigma \cup D$, which can be computed by means of a *forward-chaining* (aka *materialisation-based*) algorithm. The answer set $\mathsf{cert}(Q, \Sigma, D)$ for an arbitrary conjunctive query Q then coincides with $\mathsf{cert}(Q, \emptyset, D')$.

3. CORE TECHNICAL APPROACH

Given an OWL 2 ontology $\mathcal{O}$, our goal is to transform $\mathcal{O}$ into an OWL 2 RL ontology $\mathcal{O}'$ and (possibly) a data set $D'_{\mathcal{O}}$ such that, for any data set D and any query Q:

1. $\mathsf{cert}(Q, \mathcal{O}, D) \subseteq \mathsf{cert}(Q, \mathcal{O}', D \cup D'_{\mathcal{O}})$; and
2. $\mathsf{cert}(Q, \mathcal{O}', D \cup D'_{\mathcal{O}}) \setminus \mathsf{cert}(Q, \mathcal{O}, D)$ is "small".

As we show later on, the data set $D'_{\mathcal{O}}$ is only required if $\mathcal{O}$ contains certain kinds of constructs.

There is a tradeoff between the tightness of the upper bound (the size of $\mathsf{cert}(Q, \mathcal{O}', D \cup D'_{\mathcal{O}}) \setminus \mathsf{cert}(Q, \mathcal{O}, D)$) and the efficiency with which $\mathsf{cert}(Q, \mathcal{O}', D \cup D'_{\mathcal{O}})$ can be computed. In our approach, $\mathcal{O}'$ and $D'_{\mathcal{O}}$ are easy to compute (via a linear-time transformation), and $\mathsf{cert}(Q, \mathcal{O}', D \cup D'_{\mathcal{O}})$ can be efficiently computed using an OWL 2 RL reasoner.

To transform the ontology $\mathcal{O}$ into $\mathcal{O}'$ and $D'_{\mathcal{O}}$, we proceed as follows (see Figure 2 for a schematic representation):

1. Transform $\mathcal{O}$ into a set $\Sigma_{\mathcal{O}}$ of datalog$^{\pm,\vee}$ rules such that $\mathsf{cert}(Q, \mathcal{O}, D) = \mathsf{cert}(Q, \Sigma_{\mathcal{O}}, D)$ for any query Q (in the vocabulary of $\mathcal{O}$) and any data set D.

[3]Our definition of datalog allows conjunctions in the head, which is not allowed in the standard, but the rules can be equivalently split into multiple rules with atomic heads.

2. Transform $\Sigma_{\mathcal{O}}$ into a set $\Xi(\Sigma_{\mathcal{O}})$ of datalog rules by eliminating disjunctions and existential quantifiers, and such that for every query Q and data set D, we have $\mathsf{cert}(Q, \Sigma_{\mathcal{O}}, D) \subseteq \mathsf{cert}(Q, \Xi(\Sigma_{\mathcal{O}}), D)$.

3. Transform $\Xi(\Sigma_{\mathcal{O}})$ into an OWL 2 RL ontology $\mathcal{O}'$ and a data set $D'_{\mathcal{O}}$ such that, for every query Q and data set D, we have $\mathsf{cert}(Q, \Xi(\Sigma_{\mathcal{O}}), D) \subseteq \mathsf{cert}(Q, \mathcal{O}', D \cup D'_{\mathcal{O}})$.

Step 1 is an answer-preserving transformation from OWL 2 to datalog$^{\pm,\vee}$ rules, which can then be conveniently over-approximated in a weaker logic in the crucial second step. Step 3 is a transformation from datalog to OWL 2 RL which is answer-preserving in most (but not all) cases.

Given that $\mathcal{O}'$ is an OWL 2 RL ontology, we can use any reasoner that is sound for OWL 2 and complete for OWL 2 RL to compute a lower bound answer (using $\mathcal{O}$) and an upper bound answer (using $\mathcal{O}' \cup D'_{\mathcal{O}}$) for any given query Q and data set D. More precisely, if $\mathsf{rl}(Q, \mathcal{O}, D)$ is the query answer computed by such a reasoner, then we have:

$$\mathsf{rl}(Q, \mathcal{O}, D) \subseteq \mathsf{cert}(Q, \mathcal{O}, D) \subseteq \mathsf{rl}(Q, \mathcal{O}', D \cup D'_{\mathcal{O}}) \text{ for all } Q, D$$

We next describe the transformations in steps 1–3, and illustrate them with the example ontology $\mathcal{O}_{\mathsf{ex}}$ in Figure 4.

3.1 From OWL 2 to Datalog$^{\pm,\vee}$

The first step is to transform the OWL 2 ontology $\mathcal{O}$ into a set $\Sigma_{\mathcal{O}}$ of datalog$^{\pm,\vee}$ rules. For this, we first transform $\mathcal{O}$ into the normal form given in Definition 1. Let $\mathsf{ar}(R, x, y)$ be defined as follows for each role R occurring in $\mathcal{O}$:

$$\mathsf{ar}(R, x, y) = \begin{cases} S(y, x) & \text{if } R \text{ inverse of the atomic role } S. \\ R(x, y) & \text{if } R \text{ atomic} \end{cases}$$

Then, $\Sigma_{\mathcal{O}}$ contains the following datalog$^{\pm,\vee}$ rules for each axiom in the normalisation of $\mathcal{O}$:

- $\mathsf{lhs}(C) \rightarrow \mathsf{rhs}(C)$ for $\top \sqsubseteq C$, where $\mathsf{lhs}(C)$ and $\mathsf{rhs}(C)$ are as given in Figure 3;
- $\mathsf{ar}(R, x, y) \rightarrow \mathsf{ar}(S, x, y)$ for $R \sqsubseteq S$;
- $\mathsf{ar}(R, x, y) \wedge \mathsf{ar}(S, y, z) \rightarrow \mathsf{ar}(T, x, z)$ for $R \circ S \sqsubseteq T$; and
- $\mathsf{ar}(R, x, y) \wedge \mathsf{ar}(S, x, y) \rightarrow \bot$ for $R \sqcap T \sqsubseteq \bot$.

The obtained set $\Sigma_{\mathcal{O}}$ of datalog$^{\pm,\vee}$ rules is equivalent to the normalisation of $\mathcal{O}$, and hence it is a *conservative extension* of $\mathcal{O}$; that is, the models of $\Sigma_{\mathcal{O}}$ are obtained by extending those of $\mathcal{O}$ with the interpretation of any new predicates introduced during normalisation. Thus, $\Sigma_{\mathcal{O}}$ preserves the answers to all queries using the vocabulary of $\mathcal{O}$ [3]. The transformation of our example ontology $\mathcal{O}_{\mathsf{ex}}$ into datalog$^{\pm,\vee}$ rules $\Sigma_{\mathcal{O}_{\mathsf{ex}}}$ is also shown in Figure 4. Note that $\Sigma_{\mathcal{O}_{\mathsf{ex}}}$ is extended with a new unary predicate Aux.

3.2 From Datalog$^{\pm,\vee}$ to Datalog

Next, we transform the datalog$^{\pm,\vee}$ rules $\Sigma_{\mathcal{O}}$ into a set of datalog rules $\Xi(\Sigma_{\mathcal{O}})$ such that $\Xi(\Sigma_{\mathcal{O}}) \models \Sigma_{\mathcal{O}}$, and thus for each query Q and data set D, $\mathsf{cert}(Q, \Sigma_{\mathcal{O}}, D) \subseteq \mathsf{cert}(Q, \Xi(\Sigma_{\mathcal{O}}), D)$. This transformation is performed in two steps:

1) Rewrite each datalog$^{\pm,\vee}$ rule r into a set of datalog$^{\pm}$ rules by transforming disjunctions in the head of r into conjunctions, and splitting the resulting conjunctions into multiple datalog$^{\pm}$ rules. This is a standard "naive" technique for approximating disjunction, and was used, e.g., in the SCREECH reasoner [33]. More sophisticated strategies will be discussed later on.

2) Transform the resulting datalog$^{\pm}$ rules into datalog rules by using fresh individuals to Skolemise existentially quantified variables. Our transformation is based on the transformation from datalog$^{\pm}$ into datalog used in recent work for a rather different purpose, namely to check *chase termination* when applied to datalog$^{\pm}$ rules [4].

We next formally define the transformation $\Xi(\cdot)$ for an arbitrary set of datalog$^{\pm,\vee}$ rules; Figure 5 illustrates the application of this transformation to our running example.

DEFINITION 2. *For each datalog$^{\pm,\vee}$ rule r of the form (1) and each $1 \leq i \leq m$, let r_i be the datalog$^{\pm}$ rule*

$$r_i = B_1 \wedge ... \wedge B_n \rightarrow \exists \vec{y}_i \varphi_i(\vec{x}, \vec{y}_i)$$

and let $\varphi_i^{\wedge}(\vec{x}, \vec{y}_i)$ be defined as the conjunction of all non-inequality atoms in $\varphi_i(\vec{x}, \vec{y}_i)$.[4]

Finally, for each $1 \leq i \leq m$ and each variable $y_{ij} \in \vec{y}_i$, let c^r_{ij} be a fresh individual unique for y_{ij}, and let θ^r_i be the substitution mapping each variable $y_{ij} \in \vec{y}_i$ to c^r_{ij}. Then, $\Xi(r_i)$ is the following set of datalog rules:

$$\Xi(r_i) = \{B_1 \wedge ... \wedge B_n \rightarrow \varphi_i^{\wedge}(\vec{x}, \theta^r_i(\vec{y}_i))\} \bigcup \{c_1 \approx c_2 \rightarrow \bot \mid c_1 \not\approx c_2 \text{ occurs in } \varphi_i(\vec{x}, \theta^r_i(\vec{y}_i))\} \quad (2)$$

We finally define $\Xi(r) = \bigcup_{i=1}^{m} \Xi(r_i)$ and $\Xi(\Sigma) = \cup_{r \in \Sigma} \Xi(r)$ for Σ a set of datalog$^{\pm,\vee}$ rules.

Note that $\Xi(\Sigma)$ does not contain inequality atoms in rule heads; although such rules are allowed according to our definition of datalog, they cannot (easily) be transformed into equivalent OWL 2 RL axioms, which is our ultimate goal. For instance, the datalog$^{\pm,\vee}$ rule

$$r = B(x) \rightarrow R(x, c_1) \wedge A(c_1) \wedge R(x, c_2) \wedge A(c_2) \wedge c_1 \not\approx c_2$$

is transformed as follows, where $c_1 \approx c_2 \rightarrow \bot$ is equivalent to an OWL 2 (RL) *DifferentFrom* assertion.

$$\Xi(r) = \{B(x) \rightarrow R(x, c_1) \wedge A(c_1) \wedge R(x, c_2) \wedge A(c_2), \; c_1 \approx c_2 \rightarrow \bot\}$$

As stated in the following proposition, the proof of which is given in our Appendix A, the transformation over-approximates the datalog$^{\pm,\vee}$ rules.

PROPOSITION 1. *$\Xi(\Sigma) \models \Sigma$, for Σ an arbitrary set of datalog$^{\pm,\vee}$ rules.*

[4]Inequality atoms only occur in conjunction with other atoms in $\varphi_i(\vec{x}, \vec{y}_i)$, and hence $\varphi_i^{\wedge}(\vec{x}, \vec{y}_i)$ is well-defined.

C	$\mathsf{lhs}(C)$	$\mathsf{rhs}(C)$
A		$A(x)$
$\neg A$	$A(x)$	
$\{a\}$		$x \approx a$
$\geq n\, R.A$		$\exists y_1, \ldots, y_n \bigwedge_{1 \leq i \leq n} [\mathsf{ar}(R, x, y_C^i) \wedge A(y_C^i) \wedge \bigwedge_{i<j\leq n} y_C^i \not\approx y_C^j]$
$\geq n\, R.\neg A$		$\exists y_1, \ldots, y_n \bigwedge_{1 \leq i \leq n} [\mathsf{ar}(R, x, y_C^i) \wedge C_{\neg A}(y_C^i) \wedge \bigwedge_{i<j\leq n} y_C^i \not\approx y_C^j]$
$\exists R.\mathsf{Self}$		$\mathsf{ar}(R, x, x)$
$\neg\exists R.\mathsf{Self}$	$\mathsf{ar}(R, x, x)$	
$\forall R.A$	$\mathsf{ar}(R, x, y_C)$	$A(y_C)$
$\forall R.\neg A$	$\mathsf{ar}(R, x, y_C) \wedge A(y_C)$	
$\leq n\, R.A$	$\bigwedge_{1 \leq i \leq n+1} [\mathsf{ar}(R, x, y_C^i) \wedge A(y_C^i)]$	$\bigvee_{1 \leq i < j \leq n+1} y_C^i \approx y_C^j$
$\leq n\, R.\neg A$	$\bigwedge_{1 \leq i \leq n+1} \mathsf{ar}(R, x, y_C^i)$	$\bigvee_{1 \leq i \leq n+1} [A(y_C^i) \vee \bigvee_{1 \leq i < j \leq n+1} y_C^i \approx y_C^j]$
$C_1 \sqcup \ldots \sqcup C_n$	$\top$ if $\mathsf{lhs}(C_i)$ empty for all $1 \leq i \leq n$ $\bigwedge_{i=1}^n \mathsf{lhs}(C_i)$ otherwise	$\bot$ if $\mathsf{rhs}(C_i)$ empty for all $1 \leq i \leq n$ $\bigvee_{i=1}^n \mathsf{rhs}(C_i)$ otherwise

Note: $C_{\neg A}$ is a fresh predicate; $A(x) \wedge C_{\neg A}(x) \to \bot$ is added to the datalog$^{\pm,\vee}$ rules in the translation of $\geq n\, R.\neg A$

Figure 3: Translation of Normalised Axioms

Axioms in $\mathcal{O}_{\mathsf{ex}}$	Normalised Axioms	Datalog$^{\pm,\vee}$
$\mathsf{Student} \sqsubseteq \mathsf{Person}$	$\top \sqsubseteq \neg\mathsf{Student} \sqcup \mathsf{Person}$	$\mathsf{Student}(x) \to \mathsf{Person}(x)$
$\mathsf{RA} \sqsubseteq \mathsf{Student}$	$\top \sqsubseteq \neg\mathsf{RA} \sqcup \mathsf{Student}$	$\mathsf{RA}(x) \to \mathsf{Student}(x)$
$\mathsf{RA} \sqsubseteq \exists\mathsf{works}.\mathsf{Group}$	$\top \sqsubseteq \neg\mathsf{RA} \sqcup \exists\mathsf{works}.\mathsf{Group}$	$\mathsf{RA}(x) \to \exists y[\mathsf{works}(x, y) \wedge \mathsf{Group}(y)]$
$\mathsf{Group} \sqsubseteq \mathsf{Org}$	$\top \sqsubseteq \neg\mathsf{Group} \sqcup \mathsf{Org}$	$\mathsf{Group}(x) \to \mathsf{Org}(x)$
	$\top \sqsubseteq \neg\mathsf{Emp} \sqcup \mathsf{Person}$	$\mathsf{Emp}(x) \to \mathsf{Person}(x)$
$\mathsf{Emp} \equiv \mathsf{Person} \sqcap \exists\mathsf{works}.\mathsf{Org}$	$\top \sqsubseteq \neg\mathsf{Emp} \sqcup \exists\mathsf{works}.\mathsf{Org}$	$\mathsf{Emp}(x) \to \exists y[\mathsf{works}(x, y) \wedge \mathsf{Org}(y)]$
	$\top \sqsubseteq \mathsf{Emp} \sqcup \neg\mathsf{Person} \sqcup \forall\mathsf{works}.\neg\mathsf{Org}$	$\mathsf{Person}(x) \wedge \mathsf{works}(x, y) \wedge \mathsf{Org}(y) \to \mathsf{Emp}(x)$
$\mathsf{works} \sqsubseteq \mathsf{memberOf}$	$\mathsf{works} \sqsubseteq \mathsf{memberOf}$	$\mathsf{works}(x, y) \to \mathsf{memberOf}(x, y)$
$\mathsf{Student} \sqsubseteq \mathsf{Grad} \sqcup \mathsf{UnderGrad}$	$\top \sqsubseteq \neg\mathsf{Student} \sqcup \mathsf{Grad} \sqcup \mathsf{UnderGrad}$	$\mathsf{Student}(x) \to \mathsf{Grad}(x) \vee \mathsf{UnderGrad}(x)$
$\mathsf{func}(\mathsf{works})$	$\top \sqsubseteq \leq 1\, \mathsf{works}.\top$	$\mathsf{works}(x, y_1) \wedge \mathsf{works}(x, y_2) \to y_1 \approx y_2$
$\mathsf{Fellow} \sqsubseteq \exists\mathsf{works}.\exists\mathsf{funded}.\mathsf{Council}$	$\top \sqsubseteq \neg\mathsf{Fellow} \sqcup \exists\mathsf{works}.\mathsf{Aux}$ $\top \sqsubseteq \neg\mathsf{Aux} \sqcup \exists\mathsf{funded}.\mathsf{Council}$	$\mathsf{Fellow}(x) \to \exists y.[\mathsf{works}(x, y) \wedge \mathsf{Aux}(y)]$ $\mathsf{Aux}(x) \to \exists y[\mathsf{funded}(x, y) \wedge \mathsf{Council}(y)]$
$\mathsf{UnderGrad} \sqsubseteq \geq 3\, \mathsf{takes}.\mathsf{Course}$	$\top \sqsubseteq \neg\mathsf{UnderGrad} \sqcup \geq 3\, \mathsf{takes}.\mathsf{Course}$	$\mathsf{UnderGrad}(x) \to \exists y_1, y_2, y_3 \bigwedge_i (\mathsf{takes}(x, y_i) \wedge \mathsf{Course}(y_i) \wedge \bigwedge_{i<j\leq 3} y_i \not\approx y_j)$

Figure 4: Transforming $\mathcal{O}_{\mathsf{ex}}$ into datalog$^{\pm,\vee}$ rules $\Sigma_{\mathcal{O}_{\mathsf{ex}}}$

Proposition 1 immediately implies

$$\mathsf{cert}(Q, \Sigma_{\mathcal{O}}, D) \subseteq \mathsf{cert}(Q, \Xi(\Sigma_{\mathcal{O}}), D)$$

for an arbitrary query Q and data set D, and hence query answers w.r.t. $\Xi(\Sigma_{\mathcal{O}})$ are an upper bound to those w.r.t. $\Sigma_{\mathcal{O}}$.

Note that when $\Xi(\Sigma_{\mathcal{O}}) \cup D$ is unsatisfiable, the obtained upper bound is the trivial one for all queries, i.e., all tuples of individuals with the appropriate arity. For instance, if we extend $\mathcal{O}_{\mathsf{ex}}$ in Figure 4 with the axiom $\mathsf{Grad} \sqcap \mathsf{UnderGrad} \sqsubseteq \bot$, we obtain the $\bot$-rule $\mathsf{Grad}(x) \wedge \mathsf{UnderGrad}(x) \to \bot$ in both $\Sigma_{\mathcal{O}_{\mathsf{ex}}}$ and $\Xi(\Sigma_{\mathcal{O}_{\mathsf{ex}}})$. For $D_{\mathsf{ex}} = \{\mathsf{RA}(a)\}$ we have that $\mathcal{O}_{\mathsf{ex}} \cup D_{\mathsf{ex}}$ is satisfiable, but $\Xi(\Sigma_{\mathcal{O}_{\mathsf{ex}}}) \cup D_{\mathsf{ex}}$ is unsatisfiable. In Section 4.1 we discuss how this issue can be dealt with.

3.3 From Datalog to OWL 2 RL

The last step is to transform $\Xi(\Sigma_{\mathcal{O}})$ into an OWL 2 RL ontology $\mathcal{O}'$ and (possibly) a data set $D'_{\mathcal{O}}$.

Rules in $\Xi(\Sigma_{\mathcal{O}})$ can be of the following types (see Section 3.1, Figure 3 and Definition 2):

- R1 Rules originating from (and equivalent to) normalised role axioms $R \sqsubseteq S$, $R \circ S \sqsubseteq T$, or $R \sqcap T \sqsubseteq \bot$.
- R2 Rules $c_1 \approx c_2 \to \bot$, with c_1 and c_2 constants.
- R3 Rules originating from the transformations applied to normalised axioms of the form $\top \sqsubseteq C$.

Rules of type R1 correspond directly to OWL 2 RL axioms, which will be included in $\mathcal{O}'$. Rules of type R2 correspond to ground atoms of the form $c_1 \not\approx c_2$ (i.e., *DifferentFrom* assertions in OWL 2), which will be included in $D'_{\mathcal{O}}$.

Finally, rules of type R3 are of a very specific shape. The variables in the body are arranged in a tree-shape way, with a single root variable x, and branch variables y connected to x by atoms $R(x, y)$ or $R(y, x)$, such that each y occurs in exactly one such atom. Moreover, branch variables only occur in the rule head in atoms of the form $A(y)$ or $y \approx y'$. Rules of this form can be transformed back into OWL 2 axioms by means of the well-known rolling-up technique [15]; for example, the rule $\mathsf{Person}(x) \wedge \mathsf{works}(x, y) \wedge \mathsf{Org}(y) \to \mathsf{Emp}(x)$ can be rolled up into the axiom $\mathsf{Person} \sqcap \exists\mathsf{works}.\mathsf{Org} \sqsubseteq \mathsf{Emp}$. We formally specify this transformation in the following section.

3.3.1 Rolling up rules into OWL 2 axioms

$$
\begin{array}{rcl}
\mathsf{RA}(x) \to \exists y.[\mathsf{works}(x,y) \wedge \mathsf{Group}(y)] & \rightsquigarrow & \mathsf{RA}(x) \to \mathsf{works}(x,c_1) \wedge \mathsf{Group}(c_1) \\
\mathsf{Emp}(x) \to \exists y.[\mathsf{works}(x,y) \wedge \mathsf{Org}(y)] & \rightsquigarrow & \mathsf{Emp}(x) \to \mathsf{works}(x,c_2) \wedge \mathsf{Org}(c_2) \\
\mathsf{Student}(x) \to \mathsf{Grad}(x) \vee \mathsf{UnderGrad}(x) & \rightsquigarrow & \mathsf{Student}(x) \to \mathsf{UnderGrad}(x) \wedge \mathsf{Grad}(x) \\
\mathsf{UnderGrad}(x) \to \exists y_1,y_2,y_3 \bigwedge_i (\mathsf{takes}(x,y_i)) & \rightsquigarrow & \mathsf{UnderGrad}(x) \to \bigwedge_{i=3}^{5} (\mathsf{takes}(x,c_i) \wedge \mathsf{Course}(c_i)) \\
\wedge \bigwedge_{i<j\leq 3} y_i \not\approx y_j) & & c_i \approx c_j \to \bot \text{ for different } i \text{ and } j \\
 & & \textbf{Note: } c_1,\ldots,c_5 \text{ are fresh individuals}
\end{array}
$$

Figure 5: Transforming $\Sigma_{\mathcal{O}_{\mathsf{ex}}}$ into $\Xi(\Sigma_{\mathcal{O}_{\mathsf{ex}}})$. Only the rules that are changed by the transformation are shown.

Given a rule r of type R3, the variables occurring in r are divided into the *root variable* x, and a set of *branch variables* y, such that r satisfies the following properties, where A is a unary predicate, R is a binary predicate, c is a constant, and y, y' are branch variables:

- the body is either $\top$, or a conjunction of atoms of the form $A(x)$, $R(x,x)$, $R(x,y)$, $R(y,x)$, or $A(y)$.
- the head is either $\bot$ or a conjunction of atoms of the form $A(x)$, $R(x,x,)$, $x \approx c$, $A(y)$, $A(c)$, $R(x,c)$, $R(c,x)$, and $y \approx y'$;
- each branch variable y occurs in exactly one body atom $R(x,y)$ or $R(y,x)$; also, each constant c occurs in at most one atom $R(x,c)$ or $R(c,x)$; and
- if $y \approx y'$ occurs in the head, then y and y' occur in body atoms $R(x,y)$ or $R(y,x)$ and $R(x,y')$ or $R(y',x)$.

A rule of this form can be transformed into OWL 2 by exploiting the rolling up technique. There is, however, a technical issue related to the fresh Skolem constants in $\Xi(\Sigma_{\mathcal{O}})$. In particular, the rule $\mathsf{RA}(x) \to \mathsf{works}(x,c_1) \wedge \mathsf{Group}(c_1)$ in our running example does not directly correspond to an OWL 2 axiom. This issue can be addressed by introducing fresh roles; the above rule can be transformed into the following three OWL 2 axioms, where $S^{\mathsf{Group}}_{\mathsf{works}}$ is a fresh role:

$$\mathsf{RA} \sqsubseteq \exists S^{\mathsf{Group}}_{\mathsf{works}}.\{c_1\} \quad \exists (S^{\mathsf{Group}}_{\mathsf{works}})^-.\top \sqsubseteq \mathsf{Group} \quad S^{\mathsf{Group}}_{\mathsf{works}} \sqsubseteq \mathsf{works}$$

We are now ready to define the transformation. Note that, for simplicity, this transformation has been presented in such a way that the axiom might contain redundancies; in practice such redundancies would, of course, be eliminated.

Each atom $\alpha \in \mathsf{body}(r)$ is transformed into a concept C^α as follows, with x is the root variable of r, and y is a branch variable:

$$C^\alpha = \begin{cases} \top & \text{if } \alpha = \top; \\ A & \text{if } \alpha = A(x); \\ \exists R.\mathsf{Self} & \text{if } \alpha = R(x,x); \\ \exists R.\top & \text{if } \alpha = R(x,y); \\ \exists R^-.\top & \text{if } \alpha = R(y,x); \\ \exists R.A & \text{if } \alpha = A(y) \text{ and } R(x,y) \in \mathsf{body}(r); \\ \exists R^-.A & \text{if } \alpha = A(y) \text{ and } R(y,x) \in \mathsf{body}(r); \end{cases}$$

Each atom $\beta \in \mathsf{head}(r)$ is transformed into a concept C^β as follows, with x the root variable of r, y and y' branch variables, c a constant, and S^A_R, $S^A_{R^-}$ fresh roles:

$$C^\beta = \begin{cases} \bot & \text{if } \beta = \bot; \\ A & \text{if } \beta = A(x); \\ \exists R.\mathsf{Self} & \text{if } \beta = R(x,x); \\ \{c\} & \text{if } \beta = x \approx c; \\ \forall R.A & \text{if } \beta = A(y) \text{ and } R(x,y) \in \mathsf{body}(r); \\ \forall R^-.A & \text{if } \beta = A(y) \text{ and } R(y,x) \in \mathsf{body}(r); \\ \exists S^A_R.\{c\} & \text{if } \beta = A(c) \text{ and } R(x,c) \in \mathsf{head}(r); \text{ or} \\ & \text{if } \beta = R(x,c) \text{ and } A(c) \in \mathsf{head}(r); \\ \exists (S^A_{R^-}).\{c\} & \text{if } \beta = R(c,x) \text{ and } A(c) \in \mathsf{head}(r); \text{ or} \\ & \text{if } \beta = A(c) \text{ and } R(c,x) \in \mathsf{head}(r); \\ \leq 1\, R.A & \text{if } \beta = y \approx y' \text{ and } R(x,y), A(y) \in \mathsf{body}(r) \\ \leq 1\, R^-.A & \text{if } \beta = y \approx y' \text{ and } R(y,x), A(y) \in \mathsf{body}(r) \end{cases}$$

For simplicity, this transformation has been presented in such a way that the axiom might contain redundant conjuncts, e.g., $\leq 1\, R.A \sqcap \leq 1\, R.A$; in practice such redundancies would, of course, be eliminated.

We can now transform r into an OWL 2 axiom $C(r)$ as follows:

$$C(r) = \prod_{\alpha \in \mathsf{head}(r)} C^\alpha \sqsubseteq \prod_{\beta \in \mathsf{body}(r)} C^\beta$$

We thus obtain an ontology $\mathcal{O}'$ with the following axioms.

- Axioms of the form $R \sqsubseteq S$, $R \circ S \sqsubseteq T$, or $R \sqcap T \sqsubseteq \bot$ obtained from the rules of type R1 in $\Xi(\mathcal{O})$.
- An axiom $C(r)$ for each rule r r of type R3 in $\Xi(\mathcal{O})$ and axioms $S^A_R \sqsubseteq R$ and $\exists (S^A_R)^-.\top \sqsubseteq A$ for each fresh role S^A_R introduced in $C(r)$, with R either atomic or an inverse role.

Finally, we obtain a data set $D'_{\mathcal{O}}$ containing a ground inequality atom for each rule of type R2 in $\Xi(\mathcal{O})$.

Clearly, $\mathcal{O}' \cup D'_{\mathcal{O}}$ is a conservative extension of $\Xi(\Sigma_{\mathcal{O}})$, and hence query answers are preserved for arbitrary queries and data sets in the vocabulary of $\Xi(\Sigma_{\mathcal{O}})$.

3.3.2 Eliminating non-RL axioms

Unfortunately, $\mathcal{O}'$ might not be an OWL 2 RL ontology as it might contain the following kinds of non-RL axioms: *(i)* axioms containing the Self construct; *(ii)* axioms of the form $C \sqsubseteq \{a\}$ for $\{a\}$ a nominal concept; and *(iii)* axioms having $\top$ as the left-hand-side concept.

These kinds of axiom were excluded from OWL 2 RL due to specific design choices, rather than to inherent limitations of materialisation-based reasoning techniques; in fact, the

OWL 2 RL/RDF rules could be trivially extended to deal with such non-RL axioms. Furthermore, axioms of the kind above are rare in realistic ontologies, and none of the ontologies we used in our evaluation contained any such axiom.

If necessary, however, non-RL axioms can be eliminated by applying to $\mathcal{O}'$ and $D'_{\mathcal{O}}$ the transformations described next in the given order.

1. Replace each occurrence of a concept $\exists R.\mathsf{Self}$ on the l.h.s. of an axiom with $\exists R.\top$; and replace each axiom of the form $C \sqsubseteq \exists R.\mathsf{Self}$ with the axioms $C \sqsubseteq \{a\}$ and $C \sqsubseteq \exists R.\{a\}$ with $\{a\}$ a fresh individual.
2. Replace each axiom of the form $C \sqsubseteq \{a\}$ with an axiom $C \sqsubseteq A_a$, where A_a is a fresh concept; define a fresh atomic role P_a as inverse functional and add the axiom $A_a \sqsubseteq \exists P_a.\{a\}$; and extend $D'_{\mathcal{O}}$ with the assertion $A_a(a)$.
3. Replace each axiom of the form $\top \sqsubseteq C$ with $\mathsf{TOP} \sqsubseteq C$, where TOP is a fresh atomic concept; add axioms $A \sqsubseteq \mathsf{TOP}$, $\{a\} \sqsubseteq \mathsf{TOP}$, $\exists R.\top \sqsubseteq \mathsf{TOP}$ and $\exists R^-.\top \sqsubseteq \mathsf{TOP}$ for each atomic concept A, nominal $\{a\}$ and each role R in the ontology; and if no nominal occurs in the ontology, add the axiom $\{c\} \sqsubseteq \mathsf{TOP}$, with c a fresh individual.

These transformations could lead to additional answers to certain queries and data sets. For example, if we apply them to $\mathcal{O}' = \{\exists R.\mathsf{Self} \sqsubseteq A\}$ to obtain $\mathcal{O}'' = \{\exists R.\top \sqsubseteq A\}$ and consider $D = \{A(a), R(a,b)\}$ and $Q(x) = A(x)$, we have $\mathsf{cert}(Q, \mathcal{O}', D) = \emptyset$, whereas $\mathsf{cert}(Q, \mathcal{O}'', D) = \{a\}$.

4. ADDITIONAL CONSIDERATIONS

We next discuss some issues related to the second step in our approach, namely the transformation $\Xi(\cdot)$ from datalog$^{\pm,\vee}$ rules into datalog rules.

4.1 Dealing with Unsatisfiability

As mentioned in Section 3.2, the union of a data set D with the rules in $\Xi(\Sigma_{\mathcal{O}})$ can be unsatisfiable, even when $\Sigma_{\mathcal{O}} \cup D$ is satisfiable. This issue can be addressed by removing all $\bot$-rules from $\Xi(\Sigma_{\mathcal{O}})$, which ensures satisfiability for any D.

This is not possible without losing completeness if $\Sigma_{\mathcal{O}} \cup D$ is unsatisfiable. If $\Sigma_{\mathcal{O}} \cup D$ is satisfiable, however, $\bot$-rules intuitively do not matter because $\Xi(\cdot)$ strengthens disjunctions in $\Sigma_{\mathcal{O}}$ into conjunctions; hence, all ground atoms entailed by $\Sigma_{\mathcal{O}} \cup D$ are also entailed by $\Xi(\Sigma_{\mathcal{O}}) \cup D$ even after dispensing with the $\bot$-rules. These intuitions are formalised as follows.

THEOREM 1. *Let Σ be a set of datalog$^{\pm,\vee}$ rules, and let $\Xi_\bot(\Sigma)$ be all the $\bot$-rules in $\Xi(\Sigma)$. Then, the following condition holds for each data set D and each query Q: if $\Sigma \cup D$ is satisfiable, then $\mathsf{cert}(Q, \Sigma, D) \subseteq \mathsf{cert}(Q, \Xi(\Sigma) \setminus \Xi_\bot(\Sigma), D)$.*

The proof of the theorem is rather technical, and is deferred to our Appendix B. The idea behind the proof is, however, quite simple, and can be explained with an example.

EXAMPLE 1. *Let Σ and D be as follows:*

$$\Sigma = \{A(x) \to B(x) \vee C(x), A(x) \to D(x) \vee E(x), \\ B(x) \to \bot, C(x) \wedge D(x) \to \bot\}$$
$$D = \{A(a), C(b)\}$$

Theorem 1 applies because $\Sigma \cup D$ is satisfiable. Given

$$\Xi(\Sigma) \setminus \Xi_\bot(\Sigma) = \{A(x) \to B(x) \wedge C(x), \\ A(x) \to D(x) \wedge E(x)\}$$

we need to show that $\mathsf{cert}(Q, \Sigma, D) \subseteq \mathsf{cert}(Q, \Xi(\Sigma) \setminus \Xi_\bot(\Sigma), D)$ for an arbitrary query Q. Because $\Sigma \cup D$ is satisfiable, there exists a (Herbrand) model $\mathcal{J}$ satisfying it, say

$$\mathcal{J} = \{A(a), C(a), E(a), C(b), E(b)\}$$

Pick an arbitrary Q (say, $Q(x) = E(x)$) and an individual (say b) such that $b \notin \mathsf{cert}(Q, \Xi(\Sigma) \setminus \Xi_\bot(\Sigma), D)$. Then, there must exist a (Herbrand) interpretation $\mathcal{I}$ such that

$$\mathcal{I} \models \Xi(\Sigma) \setminus \Xi_\bot(\Sigma) \cup D \quad \text{and} \quad \mathcal{I} \not\models Q(b)$$

In our case, such an interpretation $\mathcal{I}$ could be

$$\mathcal{I} = \{A(a), B(a), C(a), D(a), E(a), C(b)\}$$

Then, we can show that the (Herbrand) interpretation $\mathcal{I} \cap \mathcal{J}$ satisfies $\Sigma \cup D$, but it does not satisfy $Q(b)$, which implies $b \notin \mathsf{cert}(Q, \Xi(\Sigma) \setminus \Xi_\bot(\Sigma), D)$, as required by the theorem.

In practice, checking the satisfiability of $\mathcal{O} \cup D$, which is equisatisfiable with $\Sigma_{\mathcal{O}} \cup D$, is easier than query answering, and even if it is impractical to check the satisfiability of $\mathcal{O} \cup D$ using an OWL 2 reasoner, e.g., if D is very large, we can still compute an upper bound "modulo satisfiability".

4.2 Transformation of Disjunctions

If $\Xi(\Sigma_{\mathcal{O}}) \cup D$ is satisfiable for a data set D, we can weaken $\Xi(\Sigma_{\mathcal{O}})$ from Definition 2 such that $\Sigma_{\mathcal{O}}$ is still entailed. In particular, when transforming a rule in $\Sigma_{\mathcal{O}}$ into datalog by replacing disjunction with conjunction, it suffices to keep only one of the conjuncts. For example, given the transformation of $A(x) \to B(x) \vee C(x)$ into $A(x) \to B(x)$ and $A(x) \to C(x)$, we can discard either of the resulting datalog rules in $\Xi(\Sigma_{\mathcal{O}})$. Each choice might result in a different upper bound. In practice we could use multiple versions of $\mathcal{O}'$ resulting from different choices to try to obtain a tighter bound, or we could make a heuristic choice of rules to retain; e.g., it makes sense to choose the rule with head predicate that appears least frequently in the bodies of other rules.

Choosing disjuncts instead of taking the conjunction of all of them is, however, incompatible with removing $\bot$-rules, and hence with Theorem 1. Consider Σ and D in Example 1 and $Q(x) = C(x)$; if $A(x) \to B(x) \vee C(x)$ is approximated to $A(x) \to B(x)$, we have $a \in \mathsf{cert}(Q, \Sigma, D)$ but $a \notin \mathsf{cert}(Q, \Xi(\Sigma) \setminus \Xi_\bot(\Sigma), D)$ and query answers are lost.

5. EXPERIMENTS

We have implemented our approach in Java and used Oracle's native OWL 2 RL reasoner in Oracle Database Release 11.2.0.3 as an OWL 2 RL reasoner. The testing machine has a dual quad core (Intel Xeon E5620) CPU, 5 SATA disks, and 40GB RAM with the operating system Linux 2.6.18.

Table 1: Statistics for data sets

Data	DL	Horn	Existential	Classes	Properties	Axioms	Individuals	Data Set
LUBM(n)	$\mathcal{SHI}$	Yes	8	43	32	93	$1.7 \times 10^4 n$	$10^5 n$
GEN-UOBM(n)	$\mathcal{SHIN}$	No	24	113	44	188	$2.5 \times 10^4 n$	$2 \times 10^5 n$
FLY	$\mathcal{SRI}$	Yes	8,396	7,533	24	144,407	1,606	6,308

5.1 Test Data

In our experiments, we have used the ontologies and data sets described next. More detailed statistics are given in Table 1.

Lehigh University Benchmark. The Leigh University Benchmark (LUBM) ontology [10] describes the organisation of universities and academic departments. Although the LUBM ontology is quite simple, it is not within the OWL 2 RL profile, as it captures existentially quantified knowledge. LUBM comes with a predefined data set generator, which can be used to test the ability of systems to handle data sets of varying size. We denote with LUBM(n) the LUBM dataset generated for n universities.

University Ontology Benchmark. The University Ontology Benchmark (UOBM) is an extension of LUBM [21] with a more complex ontology, which also contains disjunctive axioms and negation. UOBM provides three different data sets (for one, five and ten universities); in contrast to LUBM, no generator of data sets of varying size is provided for UOBM. To provide a more comprehensive evaluation, we have implemented a data generator for UOBM[5] that replicates the design of LUBM's generator. Data produced by our generator differs in several ways from the default UOBM data. This is because the data in UOBM's default data sets is skewed in what we believe are rather strange ways; for example, students in the UOBM data sets are much more likely to be connected via the isFriendOf relation to faculty members than to other students. Our generator does not replicate this skewing, and thus produces what we believe is more "realistic" data. We denote with GEN-UOBM(n) the generated UOBM data set for n universities.

Fly Anatomy (FLY). This realistic and complex ontology describing the anatomy of flies includes a data set with more than $1,000$ manually created individuals. This ontology is rich in existentially quantified knowledge and hence contains a relatively small number of OWL 2 RL axioms.

We have used two kinds of queries in our experiments.

Standard Queries. LUBM and UOBM come with 14 and 15 standard queries, respectively. Since UOBM extends LUBM, we also adapted the 14 LUBM queries to UOBM. For FLY, we have used 5 realistic queries provided by the biologists who are developing the ontology.

Synthetic Queries. We have used the system SyGENiA [6, 18] to generate synthetic queries for LUBM and UOBM and obtained 78 queries for LUBM, and 198 for UOBM (the larger number reflecting its more complex structure).

[5]`http://www.cs.ox.ac.uk/isg/tools/UOBMGenerator/`

Table 2: Synthetic LUBM queries with non-matching bounds. Upper bound is tight in all cases.

Query	Q_3	Q_{51}	Q_{67}	Q_{69}
Lower Bound	540	0	540	0
Upper Bound	1087	547	1087	547

5.2 Tightness of the Upper Bound

Results for LUBM(1). Lower and upper bounds coincide for each of the 14 LUBM standard queries and the LUBM(1) data set. This implies that Oracle's reasoner is complete for each of these queries (and the given data set), even if the ontology contains axioms outside OWL 2 RL. As to the synthetic queries, lower and upper bounds coincided in all but 4 cases (see Table 2). For these 4 queries, we used the OWL 2 reasoner HermiT to compute the exact answers, and found the upper bound to be tight in all cases.

Results for GEN-UOBM(1). Lower and upper bounds for the 15 UOBM standard queries and GEN-UOBM(1) are given in Table 3. We found matching bounds for 4 queries. For the remaining ones, the upper bound was significantly smaller than the trivial upper bound; also, by using HermiT, we determined that the lower bound was tight for 9 queries, and in the remaining 2 cases neither of the bounds was tight.

Regarding the 14 LUBM modified queries (see Table 4), we obtained matching bounds for 8 of them. For 5 of the remaining 6 queries, the lower bound was tight and the gap between bounds was typically small. For query Q_4, however, the lower bound is still tight but the gap is much larger. However, the query has a large number of answer variables, and hence a huge trivial upper bound, so the upper bound can still be considered a good approximation.

Finally, concerning the synthetic queries, we obtained matching bounds for 101(51%) of them. Figure 6 illustrates the typical size of the gap between the lower bound (LB) and upper bound (UB) answer sets, relative to the size of LB; it shows the quotient of the number of answer tuples in the gap between bounds over the number of answer tuples in the lower bound, i.e., $\frac{|\mathsf{UB} \setminus \mathsf{LB}|}{|\mathsf{LB}|}$.[6] Quotient values are presented in intervals on the X axis, and the Y axis represents the number of queries that fell within each interval; for example, we can see that for 46 queries, $\mathsf{UB} \setminus \mathsf{LB}$ contained only 10%–20% of the number of answer tuples in LB. This suggests the potential of our technique as an optimisation that efficiently identifies a small number of candidate answer tuples, which can be checked using an OWL 2 reasoner; even in the worst case, where the upper bound is almost 13 times larger than the lower bound, we have ruled out more than 99.9% of the possible answer tuples compared to the trivial upper bound.

[6]Excluding three cases where the lower bound is empty and the upper bound non-empty.

Table 3: Standard queries for UOBM

Query	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}
Lower Bound	21	2,465	581	292	235	991	0	376	1,298	8	2,416	50	0	6,271	0
Upper Bound	21	2,465	581	603	235	1,008	50	455	2,528	191	8,852	1,027	455	12,782	455
Gap	0	0	0	311	0	17	50	79	1,230	183	6,436	977	455	6,511	455
Exact Answers	21	2,465	581	292	235	991	0	376	1,298	8	2,416	50	416	6,535	0

Table 4: Modified LUBM queries for UOBM with non-matching bounds. Lower bound is tight.

Query	Q_1	Q_4	Q_5	Q_9	Q_{12}	Q_{13}
Lower Bound	0	5	648	317	41	991

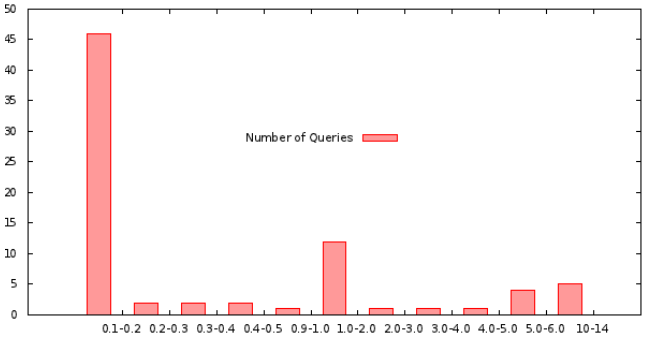


Figure 6: Synthetic UOBM queries. X axis is $\frac{|UB \setminus LB|}{|LB|}$; Y axis is the number of queries falling in each interval on the X axis.

Results for FLY. The lower and upper bounds for each of the five realistic queries are presented in Table 5. As can be seen, the lower and upper bounds coincide in Q_3, and the lower bound answers were empty for the remaining four cases. This is because the ontology includes many axioms that are outside the OWL 2 RL profile, and in particular many existential restrictions. We were able to confirm using HermiT that the upper bounds are tight for all these queries.

5.3 Scalability Tests

To test the scalability of upper bound computation using Oracle's reasoner, we have conducted experiments using LUBM and UOBM data sets of increasing size (1, 5, 10, 100 universities for LUBM and UOBM, and 1,000 universities for LUBM). We also report computation times for FLY.

Test for LUBM. Results for all the standard queries and generated queries are summarised in Figure 7(a); in the figure, materialisation time refers to the total time for computing the saturation for each data set and querying time refers to the average query answering time for each query. We can observe that query answering times and scalability behaviour is very similar for lower and upper bound computation. Fully-fledged OWL 2 reasoners are much slower, even for the smallest data sets; for LUBM(1), HermiT required 7,684 seconds to compute the exact answers to one of the queries with matching lower and upper bounds.

Table 5: Realistic queries for FLY. Upper bound is tight in all cases.

Query	Q_1	Q_2	Q_3	Q_4	Q_5
Lower Bound	0	0	28	0	0
Upper Bound	803	342	28	25	518

Test for GEN-UOBM. Results for both standard and generated queries are given in Figure 7(b). In this case, the materialisation time is higher for the upper bound than that for the lower bound because of the increased number of materialised triples. The time to answer queries also increases significantly, but is in line with the increased size of the answer. For example, the lower bound for generated Query 195 on UOBM(10) contains 132,411 answer tuples, whereas the upper bound contains 1,961,095 answer tuples. Although less efficient than lower bound computation, upper bound computation significantly outperforms HermiT, and the lower and upper bounds coincide for 9 out of 14 queries. Upper bound computation required less than 2 seconds for *all* standard queries w.r.t. UOBM(1); HermiT, in comparison, failed to compute the answer to one of the standard queries (Query 6)[7], even when given a 24h timeout. We also used HermiT to check tuples in the gap between the lower bound and upper bound for this query, which took only 1 hour. This illustrates the potential of upper and lower bound answers in optimising the computation of exact answers.

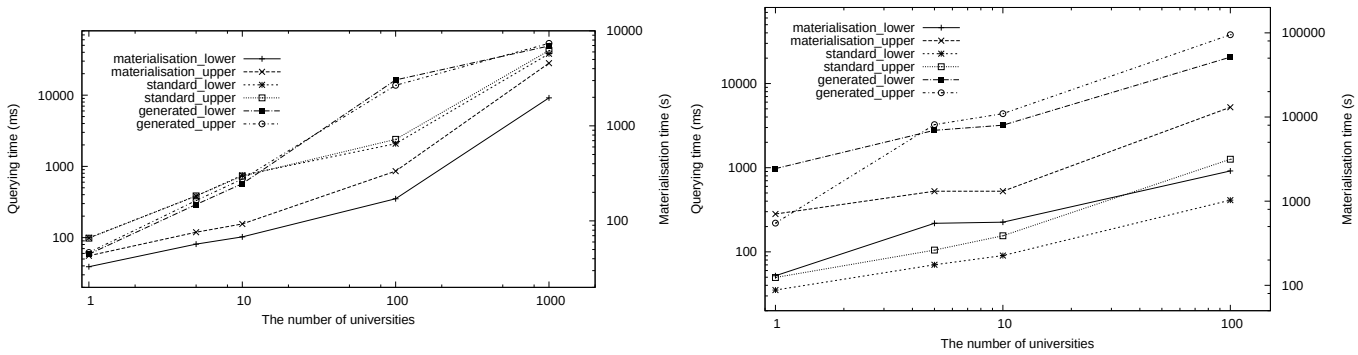
Test for FLY. Oracle's reasoner required 164s and 493s respectively to compute the lower and upper bound materialisation, and to answer all queries. The query answering time was negligible compared to materialisation time.

6. RELATED WORK

Our work is related to *theory approximation*, which was first described in the seminal paper by Kautz and Selman [31]. The idea in theory approximation is to approximate a logical theory $\mathcal{T}$ by two theories $\mathcal{T}_{lb}$ (the *model lower bound*) and $\mathcal{T}_{ub}$ (the *model upper bound*) such that $\mathcal{T}_{lb} \models \mathcal{T} \models \mathcal{T}_{ub}$, both $\mathcal{T}_{lb}$ and $\mathcal{T}_{ub}$ are in a "more tractable" language than $\mathcal{T}$, and $\mathcal{T}_{lb}$ and $\mathcal{T}_{ub}$ are "as close as possible" to $\mathcal{T}$. Kautz and Selman studied this problem for $\mathcal{T}$ in propositional logic and the bounds expressed in its Horn fragment. Del Val [8] studied the problem for first-order logic. This line of research has focused mostly on the computation of the "best" model upper bounds; however, we focus on query answers rather than models and hence our upper bounds correspond to *model lower bounds*, which have received little attention.

The idea of transforming the ontology, data, and/or query to obtain upper bounds to query answers has been already

[7] $Q(x) \leftarrow \mathsf{hasAlumnus}(\mathsf{Univ0}, x) \wedge \mathsf{Person}(x)$



(a) Time for LUBM

(b) Time for UOBM

Figure 7: Scalability tests

Table 6: Comparison between different systems

System	Source	Target	Independence		Time	Query
			Data	Query		
Screech [33]	$\mathcal{SHIQ}$	Datalog	NO	YES	exponential	SPARQL
Quill [28]	OWL DL	DL-Lite	NO	NO	exponential	CQ
[19]	$\mathcal{SHI}$	DL-Lite	YES	YES	exponential	SPARQL
Ours	OWL 2	OWL 2 RL	YES	YES	polynomial	CQ

explored in previous work. Table 6 summarises the main differences between our approach and the systems presented in [33, 28, 19], which we next explain in more detail.

The Screech system [33] uses KAON2 [17] to transform an ontology into a disjunctive datalog program such that answers to SPARQL queries are preserved, and then approximates the resulting disjunctive program into a datalog program by transforming disjunctions into conjunctions. The transformation of the ontology (which is delegated to KAON2) requires exponential time (and may also be of exponential size) in the size of the input ontology. This exponential blow-up means that, in practice, KAON2 may be unable to process large or complex ontologies; for example, KAON2 was reported to fail on the DOLCE ontology [24]. Finally, due to the dependency on KAON2, Screech can only deal with the subset of OWL 2 corresponding to the $\mathcal{SHIQ}$ DL, and is guaranteed to compute an upper bound only for SPARQL queries; in contrast our approach applies to all of OWL 2 as well as to more general query languages.

The Quill system transforms both the ontology $\mathcal{O}$ and query Q to compute an upper bound [28]. In this case, the target language for approximation is DL-Lite (a.k.a. OWL 2 QL), instead of OWL 2 RL. Quill first transforms Q and adds axioms $\mathcal{O}_Q$ to $\mathcal{O}$ based on this transformation. Then, Quill computes as an approximation OWL 2 QL axioms entailed by $\mathcal{O} \cup \mathcal{O}_Q \cup D$, with D the input data set. Each entailment test requires the use of a fully-fledged OWL reasoner, which can be expensive; also, the required entailments need to be recomputed for each query and each data set.

Kaplunova et al. [19] approximate an ontology $\mathcal{O}$ into an OWL 2 QL ontology $\mathcal{O}'$ to provide an upper bound to queries in SPARQL. Each axiom $C \sqsubseteq D$ in $\mathcal{O}$ is transformed into an OWL 2 QL axiom $C' \sqsubseteq D'$, where C is subsumed by C' and D' is subsumed by D (w.r.t. $\mathcal{O}$). The transformation algorithm, however, is non-deterministic and there can be exponentially many C' and D' satisfying the required properties. Furthermore, as reported in [19], it is often the case that for a given D such that $\mathcal{O} \cup D$ is satisfiable, $\mathcal{O}' \cup D$ is unsatisfiable, regardless of the choices made when computing $\mathcal{O}'$. The large degree of non-determinism means that computing $\mathcal{O}'$ can be expensive, even for small ontologies—it is reported in [19] that "it is very demanding to approximate a TBox with 499 axioms", and that they were unable to compute a coherent approximation "in reasonable time".

7. DISCUSSION

We have proposed novel techniques that allow us to exploit industrial-strength triple stores to answer queries over ontologies that are outside OWL 2 RL, thus "making the most" of state-of-the-art triple store technologies. Our techniques allow us to compute exact answers to queries in many cases. Otherwise, we can still efficiently compute an upper bound to the exact answers, which allows us to estimate the incompleteness of the triple store as well as to optimise OWL 2 reasoners by ruling out many candidate answer tuples.

The results obtained so far open many possibilities for future work. For example, we plan to develop techniques for identifying, during upper bound computation, a (hopefully small) fragment of the ontology and data set that is sufficient for checking whether the answers in the gap between

bounds are indeed answers; this fragment can then be used instead of the original ontology when checking answers in the gap using an OWL 2 reasoner.

8. REFERENCES

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

[2] A. Cali, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS*, 2010.

[3] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular reuse of ontologies: Theory and practice. *JAIR*, 31:273–318, 2008.

[4] B. Cuenca Grau, I. Horrocks, M. Krötzsch, C. Kupke, D. Magka, B. Motik, and Z. Wang. Acyclicity conditions and their application to query answering in description logics. In *KR*, 2012.

[5] B. Cuenca Grau, B. Motik, G. Stoilos, and I. Horrocks. Completeness guarantees for incomplete ontology reasoners: Theory and practice. *J. of Artificial Intelligence Research (JAIR)*, 43, 2012.

[6] B. Cuenca Grau and G. Stoilos. What to ask to an incomplete semantic web reasoner? In *IJCAI*, pages 419–476, 2011.

[7] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[8] A. Del Val. First order LUB approximations: characterization and algorithms. *Artificial Intelligence*, 162(1-2):7–48, 2005.

[9] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *WWW*, 2003.

[10] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Semantics (JWS)*, 3(2-3):158–182, 2005.

[11] V. Haarslev and R. Möller. RACER system description. *J. of Automated Reasoning (JAR)*, pages 701–705, 2001.

[12] V. Haarslev, R. Möller, and M. Wessel. Querying the semantic web with RACER+NRQL. In *ADL*, 2004.

[13] A. Harth, J. Umbrich, A. Hogan, and S. Decker. Yars2: A federated repository for querying graph structured data from the web. *The Semantic Web*, pages 211–224, 2007.

[14] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible $\mathcal{SROIQ}$. In *KR*, 2006.

[15] I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *AAAI*, 2000.

[16] J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL querying of large RDF graphs. *PVLDB*, 4(11):1123–1134, 2011.

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

[18] M. Imprialou, G. Stoilos, and B. Grau. Benchmarking ontology-based query rewriting systems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, AAAI*, 2012.

[19] A. Kaplunova, R. Möller, S. Wandelt, and M. Wessel. Towards scalable instance retrieval over ontologies. *Knowledge Science, Engineering and Management*, pages 436–448, 2010.

[20] I. Kollia, B. Glimm, and I. Horrocks. Query answering over SROIQ knowledge bases with SPARQL. In *DL*, 2011.

[21] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a complete OWL ontology benchmark. In *ESWC*, pages 125–139, 2006.

[22] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles. *W3C Recommendation*, 2009.

[23] B. Motik, P. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler, et al. OWL 2 Web Ontology Language: Structural Specification and Functional-style Syntax. *W3C recommendation*, 27:17, 2009.

[24] B. Motik and U. Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 227–241. Springer, 2006.

[25] B. Motik, R. Shearer, and I. Horrocks. Hypertableau reasoning for description logics. *J. of Artificial Intelligence Research (JAIR)*, 36(1):165–228, 2009.

[26] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113, 2010.

[27] A. Nonnengart and C. Weidenbach. Computing small clause normal forms. *Handbook of automated reasoning*, 1:335–367, 2001.

[28] J. Pan, E. Thomas, and Y. Zhao. Completeness guaranteed approximations for OWL-DL query answering. *Proc. of DL*, 477, 2009.

[29] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.

[30] K. Rohloff and R. Schantz. High-performance, massively scalable distributed systems using the mapreduce software framework: The shard triple-store. In *Programming Support Innovations for Emerging Distributed Applications*, 2010.

[31] B. Selman and H. Kautz. Knowledge compilation and theory approximation. *J. of the ACM (JACM)*, 43(2):193–224, 1996.

[32] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. Web Semantics (JWS)*, 5(2):51–53, 2007.

[33] T. Tserendorj, S. Rudolph, M. Krötzsch, and P. Hitzler. Approximate OWL-reasoning with screech. In *RR*, number 5341 in LNCS, pages 165–180, 2008.

[34] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment*, 1(1):1008–1019, 2008.

[35] Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan. Implementing an

inference engine for RDFS/OWL constructs and user-defined rules in Oracle. In *ICDE*, pages 1239–1248, 2008.

APPENDIX

A. PROOF OF PROPOSITION 1

In this section, we prove Proposition 1, namely $\Xi(\Sigma) \models \Sigma$ for Σ a set of datalog$^{\pm,\vee}$ rules.

The following proposition provides sufficient and necessary conditions for a datalog$^{\pm,\vee}$ rule to be entailed by a set of first-order sentences (the proof is rather straightforward, and can be found in [5]).

PROPOSITION 2. *Let $\mathcal{F}$ be a set of first-order sentences and r be a datalog$^{\pm,\vee}$ rule of the form (1). Then, for each substitution σ mapping the free variables of r to distinct individuals not occurring in $\mathcal{F}$ or r, we have $\mathcal{F} \models r$ iff*

$$\mathcal{F} \cup \{\sigma(B_1), \ldots, \sigma(B_n)\} \models \bigvee_{i=1}^{m} \exists \vec{y}_i.\varphi_i(\sigma(\vec{x}), \vec{y}_i)$$

Proposition 2 can then be used to show that each datalog$^{\pm,\vee}$ rule r in Σ is entailed by $\Xi(r)$, and hence $\Xi(\Sigma) \models \Sigma$.

PROOF OF PROPOSITION 1. It suffices to show that, for each rule $r \in \Sigma$ of the form (1) and each $1 \leq i \leq m$, we have $\Xi(r_i) \models r$. Let σ be a substitution mapping the free variables in r to fresh individuals; by Proposition 2, we have

$$\Xi(r_i) \models r \Leftrightarrow \Xi(r_i) \cup \{\sigma(B_1), \ldots, \sigma(B_n)\} \models \bigvee_{i=1}^{m} \exists \vec{y}_i.\varphi_i(\sigma(\vec{x}), \vec{y}_i)$$

According to the definition of $\varphi_i^{\wedge}$, we have

$$\varphi_i(\sigma(\vec{x}), \theta_i^r(\vec{y}_i)) = \varphi_i^{\wedge}(\sigma(\vec{x}), \theta_i^r(\vec{y}_i)) \wedge \bigwedge c_{ij}^r \wedge c_{ij'}^r \qquad (3)$$

where $c_{ij}^r \wedge c_{ij'}^r$ occurs in $\varphi_i(\sigma(\vec{x}), \theta_i^r(\vec{y}_i))$.

Clearly, for each inequality assertion $c_{ij}^r \wedge c_{ij'}^r$ occurring in $\varphi_i(\sigma(\vec{x}), \theta_i^r(\vec{y}_i))$ we have $c_{ij}^r \approx c_{ij'}^r \rightarrow \bot \in \Xi(r_i)$ and hence

$$\Xi(r_i) \cup \{\sigma(B_1), \ldots, \sigma(B_n)\} \models c_{ij}^r \wedge c_{ij'}^r \qquad (4)$$

Moreover, $\Xi(r_i)$ contains the rule $B_1 \wedge ... \wedge B_n \rightarrow \varphi_i^{\wedge}(\vec{x}, \theta_i^r(\vec{y}_i))$ and hence

$$\Xi(r_i) \cup \{\sigma(B_1), \ldots, \sigma(B_n)\} \models \varphi_i^{\wedge}(\sigma(\vec{x}), \theta_i^r(\vec{y}_i)) \qquad (5)$$

Combining condition (3), (4), (5), we have

$$\Xi(r_i) \cup \{\sigma(B_1), \ldots, \sigma(B_n)\} \models \varphi_i(\sigma(\vec{x}), \theta_i^r(\vec{y}_i))$$

Since θ_i^r maps variables to constants, the following conditions clearly hold by the semantics of first-order logic:

$$\begin{aligned} \varphi_i(\sigma(\vec{x}), \theta_i^r(\vec{y}_i)) &\models \exists \vec{y}_i.\varphi_i(\sigma(\vec{x}), \vec{y}_i) \\ &\models \bigvee_{i=1}^{m} \exists \vec{y}_i.\varphi_i(\sigma(\vec{x}), \vec{y}_i) \end{aligned}$$

As a result, $\Xi(r_i) \models r$. □

B. PROOF FOR THEOREM 1

In the proof of Theorem 1, we use standard Skolemisation [27] to eliminate existential quantifiers. It is well-known that the Skolemisation preserves satisfiability. We next prove that the Skolemisation also preserves certain answers for queries.

LEMMA 1. *Let $\mathcal{F}$ be a set of rectified sentences,*[8] *and let $\mathsf{sk}(\mathcal{F})$ be the formulas obtained by applying standard Skolemisation to $\mathcal{F}$. Then, the following condition holds for each data set D and each conjunctive query Q:*

$$\mathsf{cert}(Q, \mathcal{F}, D) = \mathsf{cert}(Q, \mathsf{sk}(\mathcal{F}), D)$$

PROOF OF LEMMA 1. Recall that D is a finite set of ground atoms and that Q is a formula of the form $Q(\vec{x}) = \exists \vec{y}.\varphi(\vec{x}, \vec{y})$ where φ is a conjunction of atoms. Therefore, we have the following:

$$\mathsf{sk}(\mathcal{F} \cup D \cup \{\neg Q(\vec{x})\}) = \mathsf{sk}(\mathcal{F}) \cup D \cup \{\neg Q(\vec{x})\} \qquad (6)$$

And hence, we also have the following for $\vec{a}$ a tuple of constants:

$\vec{a} \in \mathsf{cert}(Q, \mathcal{F}, D)$
iff $\mathcal{F} \cup D \models Q(\vec{a})$
iff $\mathcal{F} \cup D \cup \{\neg Q(\vec{a})\}$ is unsatisfiable
iff $\mathsf{sk}(\mathcal{F} \cup D \cup \{\neg Q(\vec{a})\})$ is unsatisfiable
iff $\mathsf{sk}(\mathcal{F}) \cup D \cup \{\neg Q(\vec{a})\}$ is unsatisfiable[due to (6)]
iff $\mathsf{sk}(\mathcal{F}) \cup D \models Q(\vec{a})$
iff $\vec{a} \in \mathsf{cert}(Q, \mathsf{sk}(\mathcal{F}), D)$

□

In this paper, we have so far used first-order logic with equality. It is well-known, that equality can be axiomatised in first-order logic without equality such that entailment of first-order logic formulas is preserved. In the proof of Theorem 1, we will exploit Herbrand's theorem, which does not hold in the presence of equality; therefore, we use the explicit axiomatisation of equality, rather than consider equality as a special predicate with a fixed interpretation. The axiomatisation of equality for a first-order signature is the following set of sentences, where f is a function and P is a predicate in the relevant signature.

$$\begin{gathered} x \approx x \\ x \approx y \rightarrow y \approx x \\ x \approx y, y \approx z \rightarrow x \approx z \\ x_1 \approx y_1, \ldots, x_n \approx y_n \rightarrow f(x_1, \ldots, x_n) \approx f(y_1, \ldots, y_n) \\ x_1 \approx y_1, \ldots, x_n \approx y_n, P(x_1, \ldots, x_n) \rightarrow P(y_1, \ldots, y_n) \end{gathered}$$

PROOF OF THEOREM 1. Consider the rules in $\Xi_{\bot}(\Sigma)$. By the definition of $\Xi(\cdot)$, we observe that each rule in $\Xi_{\bot}(\Sigma)$ satisfies one of the following: it is either in Σ, or it is of the form $c_{ij}^r \approx c_{ij'}^r \rightarrow \bot$ and hence it originates from an inequality assertion in the head of some rule $r \in \Sigma$. We can therefore obtain $\Xi(\Sigma) \setminus \Xi_{\bot}(\Sigma)$ in the following steps

[8] That is, no variable appears quantified more than once.

1. Compute $\Sigma' = \Sigma \backslash \Sigma_\perp$, where $\Sigma_\perp$ is the set of all $\perp$-rules in Σ;
2. Replace each rule $B_1 \wedge \ldots \wedge B_n \rightarrow \bigvee_{i=1}^m \varphi(\vec{x}, \vec{y}_i)$ in Σ' with the rule $B_1 \wedge \ldots B_n \rightarrow \bigvee_{i=1}^m \varphi^\wedge(\vec{x}, \vec{y}_i)$ and then replace disjunctions by conjunctions in all rules; we denote with $\Pi(\Sigma')$ the resulting set of rules

$$\begin{array}{ccc} \Sigma' & \rightsquigarrow & \Pi(\Sigma') \\ \ldots \rightarrow \bigvee_i \exists \vec{y}_i.\varphi_i(\vec{x}, \vec{y}_i) & & \ldots \rightarrow \bigwedge_i \exists \vec{y}_i.\varphi_i^\wedge(\vec{x}, \vec{y}_i) \end{array}$$

3. Replace each existentially quantified variable with a fresh Skolem constant, thus obtaining $\Xi(\Sigma) \setminus \Xi_\perp(\Sigma)$.

Clearly, we have $\Xi(\Sigma) \setminus \Xi_\perp(\Sigma) \models \Pi(\Sigma')$; hence, we have $\mathsf{cert}(Q, \Pi(\Sigma'), D) \subseteq \mathsf{cert}(Q, \Xi(\Sigma) \setminus \Xi_\perp(\Sigma), D)$ for each data set D and each query Q. Consequently, what we need to prove is that for each D and Q,

$$\Sigma \cup D \text{ satisfiable} \Rightarrow \mathsf{cert}(Q, \Sigma, D) \subseteq \mathsf{cert}(Q, \Pi(\Sigma'), D) \quad (7)$$

Consider the standard Skolemisation $\mathsf{sk}(\Sigma')$ of Σ'. Since $\Sigma_\perp$ contains no existential quantifier, we have $\mathsf{sk}(\Sigma_\perp) = \Sigma_\perp$; hence, $\mathsf{sk}(\Sigma) = \mathsf{sk}(\Sigma') \cup \Sigma_\perp$ is a Skolemisation of Σ and

$$\Sigma \cup D \text{ satisfiable} \Leftrightarrow \mathsf{sk}(\Sigma') \cup \Sigma_\perp \cup D \text{ satisfiable} \quad (8)$$

Furthermore, by Lemma 1, we have the following for each query Q and data set D:

$$\mathsf{cert}(Q, \mathsf{sk}(\Sigma') \cup \Sigma_\perp, D) = \mathsf{cert}(Q, \Sigma, D) \quad (9)$$

Since Σ' and $\Pi(\Sigma')$ share the same existential quantifiers and existentially quantified variables, we can reuse the Skolem function symbols in $\mathsf{sk}(\Sigma')$ to compute a Skolemisation $\mathsf{sk}(\Pi(\Sigma'))$ of $\Pi(\Sigma')$. By Lemma 1, we have the following for each Q and D:

$$\mathsf{cert}(Q, \mathsf{sk}(\Pi(\Sigma')), D) = \mathsf{cert}(Q, \Pi(\Sigma'), D) \quad (10)$$

Therefore, it suffices to show the following condition:

$$\begin{array}{l} \mathsf{sk}(\Sigma') \cup \Sigma_\perp \cup D \text{ satisfiable} \\ \Rightarrow \ \mathsf{cert}(Q, \mathsf{sk}(\Sigma') \cup \Sigma_\perp, D) \subseteq \mathsf{cert}(Q, \mathsf{sk}(\Pi(\Sigma')), D) \end{array} \quad (11)$$

Indeed, if (11) holds, then (8),(9) and (10) clearly imply (7).

To show (11), it is equivalent to prove the following condition for each data set D, each query Q and each tuple $\vec{a}$.

$$\begin{array}{l} \mathsf{sk}(\Sigma') \cup \Sigma_\perp \cup D \text{ satisfiable and } \mathsf{sk}(\Pi(\Sigma')) \cup D \not\models Q(\vec{a}) \\ \Rightarrow \ \mathsf{sk}(\Sigma') \cup \Sigma_\perp \cup D \not\models Q(\vec{a}) \end{array}$$

Let $\Lambda_\approx$ be the axiomatisation of equality for the signature of $\mathsf{sk}(\Sigma')$. Let D be an arbitrary data set such that $\mathsf{sk}(\Sigma') \cup \Sigma_\perp \cup D$ is satisfiable, let Q be an arbitrary query, and let $\vec{a}$ be a tuple s.t. $\mathsf{sk}(\Pi(\Sigma')) \cup \Lambda_\approx \cup D \not\models Q(\vec{a})$. Then $\mathsf{sk}(\Pi(\Sigma')) \cup \Lambda_\approx \cup D \cup \{\neg Q(\vec{a})\}$ is satisfiable. According to Herbrand's theorem, there exists a Herbrand interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathsf{sk}(\Pi(\Sigma')) \cup \Lambda_\approx \cup D \cup \{\neg Q(\vec{a})\}$. Therefore,

$$\mathcal{I} \models \mathsf{sk}(\Pi(\Sigma')) \cup \Lambda_\approx \cup D, \text{ but } \mathcal{I} \not\models Q(\vec{a}).$$

Moreover, $\mathsf{sk}(\Sigma') \cup \Sigma_\perp \cup \Lambda_\approx \cup D$ is satisfiable implies there is an Herbrand interpretation $\mathcal{J}$ s.t.

$$\mathcal{J} \models \mathsf{sk}(\Sigma') \cup \Sigma_\perp \cup \Lambda_\approx \cup D$$

Because $\mathcal{I} \not\models Q(\vec{a})$, $\mathcal{I} \cap \mathcal{J} \subseteq \mathcal{I}$ and all the atoms in Q are positive, it is trivial that $\mathcal{I} \cap \mathcal{J} \not\models Q(\vec{a})$. We will show that $\mathcal{I} \cap \mathcal{J} \models \mathsf{sk}(\Sigma') \cup \Sigma_\perp \cup \Lambda_\approx \cup D$, which means $\mathcal{I} \cap \mathcal{J}$ is a counter-model of $Q(\vec{a})$ w.r.t. $\mathsf{sk}(\Sigma') \cup \Sigma_\perp \cup \Lambda_\approx \cup D$.

Since $\mathcal{I} \cap \mathcal{J} \subseteq \mathcal{J}$ and $\mathcal{J} \models \Sigma_\perp$, we have $\mathcal{I} \cap \mathcal{J} \models \Sigma_\perp$ because $\mathcal{I} \cap \mathcal{J}$ has fewer positive atoms than $\mathcal{J}$. It is also easy to verify that $\mathcal{I} \cap \mathcal{J} \models \Lambda_\approx$. Furthermore, we next show that $\mathcal{I} \cap \mathcal{J} \models D$. Since $\mathcal{I} \models D$ and $\mathcal{J} \models D$, given that any grounded atom $\alpha \in D$,

- if α is not an inequality atom, then we have $\alpha \in \mathcal{I}$ and $\alpha \in \mathcal{J}$; thus $\mathcal{I} \cap \mathcal{J} \models \alpha$;
- if α is an inequality atom of the form $c_1 \not\approx c_2$, then we have $c_1 \approx c_2 \notin \mathcal{I}$ and $c_1 \approx c_2 \notin \mathcal{J}$; thus $\mathcal{I} \cap \mathcal{J} \models \alpha$ as well.

So $\mathcal{I} \cap \mathcal{J} \models D$. Consequently, the only remaining task is to show that $\mathcal{I} \cap \mathcal{J} \models \mathsf{sk}(\Sigma')$.

Assume by contradiction that $\mathcal{I} \cap \mathcal{J} \not\models \mathsf{sk}(\Sigma')$, i.e. there exists a rule $r \in \mathsf{sk}(\Sigma')$ and a tuple of constants $\vec{c}$ such that $\mathcal{I} \cap \mathcal{J} \not\models B_1(\vec{c}), \ldots, B_n(\vec{c}) \rightarrow \bigvee_{i=1}^m \varphi_i(\vec{c}, \vec{f_i}(\vec{c}))$. Thus, $\{B_1(\vec{c}), \ldots, B_n(\vec{c})\} \subseteq \mathcal{I} \cap \mathcal{J}$.

Since $\mathcal{I} \cap \mathcal{J} \subseteq \mathcal{I}$, we have $\{B_1(\vec{c}), \ldots, B_n(\vec{c})\} \subseteq \mathcal{I}$; furthermore, $\mathcal{I} \models \mathsf{sk}(\Pi(\Sigma')$ and hence $\mathcal{I} \models \varphi_i^\wedge(\vec{c}, \vec{f_i}(\vec{c}))$ for each $1 \leq i \leq m$. Similarly, since $\mathcal{I} \cap \mathcal{J} \subseteq \mathcal{J}$ we also have $\{B_1(\vec{c}), \ldots, B_n(\vec{c})\} \subseteq \mathcal{J}$ and since $\mathcal{J} \models \mathsf{sk}(\Sigma')$ then there exists $k \in \{1, \ldots, m\}$ such that $\mathcal{J} \models \varphi_k(\vec{c}, \vec{f_k}(\vec{c}))$.

If $\varphi_k(\vec{c}, \vec{f_k}(\vec{c}))$ does not contain inequality assertions, then $\mathcal{I} \models \varphi_k(\vec{c}, \vec{f_k}(\vec{c}))$ and hence $\mathcal{I} \cap \mathcal{J} \models \varphi_k(\vec{c}, \vec{f_k}(\vec{c}))$, which implies $\mathcal{I} \cap \mathcal{J} \models B_1(\vec{c}), \ldots, B_n(\vec{c}) \rightarrow \bigvee_{i=1}^m \varphi_i(\vec{c}, \vec{f_i}(\vec{c}))$ leading to a contradiction.

If $\varphi_k(\vec{c}, \vec{f_k}(\vec{c}))$ contains an inequality assertion, say $t_1 \not\approx t_2$ for t_1 and t_2 two terms. Since $\mathcal{J} \models \varphi_k(\vec{c}, \vec{f_k}(\vec{c}))$, we have $J \models t_1 \not\approx t_2$ and thus $t_1 \approx t_2 \notin \mathcal{J}$. As a result, we have $t_1 \approx t_2 \notin \mathcal{I} \cap \mathcal{J}$ and then $\mathcal{I} \cap \mathcal{J} \models t_1 \not\approx t_2$. Furthermore, $\mathcal{I} \models \varphi_k^\wedge(\vec{c}, \vec{f_k}(\vec{c}))$, $\mathcal{J} \models \varphi_k(\vec{c}, \vec{f_k}(\vec{c}))$ and $\varphi_k(\vec{c}, \vec{f_k}(\vec{c})) \models \varphi_k^\wedge(\vec{c}, \vec{f_k}(\vec{c}))$ implies $\mathcal{I} \cap \mathcal{J} \models \varphi_k^\wedge(\vec{c}, \vec{f_k}(\vec{c}))$. Therefore, $\mathcal{I} \cap \mathcal{J} \models \varphi_k(\vec{c}, \vec{f_k}(\vec{c}))$, which implies $\mathcal{I} \cap \mathcal{J} \models B_1(\vec{c}), \ldots, B_n(\vec{c}) \rightarrow \bigvee_{i=1}^m \varphi_i(\vec{c}, \vec{f_i}(\vec{c}))$ - a contradiction. $\square$