# Computing Datalog Rewritings Beyond Horn Ontologies*

**Bernardo Cuenca Grau**[1] and **Boris Motik**[1] and **Giorgos Stoilos**[2] and **Ian Horrocks**[1]

[1]Department of Computer Science
University of Oxford, UK

[2]School of Electrical and Computer Engineering
National Technical University of Athens, Greece

## Abstract

Rewriting-based approaches for answering queries over an OWL 2 DL ontology have so far been developed mainly for Horn fragments of OWL 2 DL. In this paper, we study the possibilities of answering queries over non-Horn ontologies using datalog rewritings. We prove that this is impossible in general even for very simple ontology languages, and even if PTIME = NP. Furthermore, we present a resolution-based procedure for $\mathcal{SHI}$ ontologies that, in case it terminates, produces a datalog rewriting of the ontology. We also show that our procedure necessarily terminates on DL-Lite$^{\mathcal{H},+}_{bool}$ ontologies—an extension of OWL 2 QL with transitive roles and Boolean connectives.

## 1 Introduction

Answering conjunctive queries (CQs) over OWL 2 DL ontologies is a computationally hard [Glimm *et al.*, 2008; Lutz, 2008], but key problem in many applications. Thus, considerable effort has been devoted to the development of OWL 2 DL fragments for which query answering is tractable in *data complexity*, which is measured in the size of the data only. Most languages obtained in this way are *Horn*: ontologies in such languages can always be translated into first-order Horn clauses. This includes the families of 'lightweight' languages such as DL-Lite [Calvanese *et al.*, 2007], $\mathcal{EL}$ [Baader *et al.*, 2005], and DLP [Grosof *et al.*, 2003] that underpin the QL, EL, and RL profiles of OWL 2, respectively, as well as more expressive languages, such as Horn-$\mathcal{SHIQ}$ [Hustadt *et al.*, 2005] and Horn-$\mathcal{SROIQ}$ [Ortiz *et al.*, 2011].

Query answering can sometimes be implemented via query rewriting: a rewriting of a query $Q$ w.r.t. an ontology $\mathcal{T}$ is another query $Q'$ that captures all information from $\mathcal{T}$ necessary to answer $Q$ over an arbitrary data set. Unions of conjunctive queries (UCQs) and datalog are common target languages for query rewriting. They ensure tractability w.r.t. data complexity, while enabling the reuse of optimised data management systems: UCQs can be answered using relational databases [Calvanese *et al.*, 2007], and datalog queries can be answered using rule-based systems such as OWLim [Bishop *et al.*, 2011] and Oracle's Semantic Data Store [Wu *et al.*, 2008]. Query rewriting algorithms have so far been developed mainly for Horn fragments of OWL 2 DL, and they have been implemented in systems such as QuOnto [Acciarri *et al.*, 2005], Rapid [Chortaras *et al.*, 2011], Presto [Rosati and Almatelli, 2010], Quest [Rodriguez-Muro and Calvanese, 2012], Clipper [Eiter *et al.*, 2012], Owlgres [Stocker and Smith, 2008], and Requiem [Pérez-Urbina *et al.*, 2010].

Horn fragments of OWL 2 DL cannot capture *disjunctive knowledge*, such as 'every student is either an undergraduate or a graduate'. Such knowledge occurs in practice in ontologies such as the NCI Thesaurus and the Foundational Model of Anatomy, so these ontologies cannot be processed using known rewriting techniques; furthermore, no query answering technique we are aware of is tractable w.r.t. data complexity when applied to such ontologies. These limitations cannot be easily overcome: query answering in even the basic non-Horn language $\mathcal{ELU}$ is co-NP-hard w.r.t. data complexity [Krisnadhi and Lutz, 2007], and since answering datalog queries is PTIME-complete, it may not be possible to rewrite an arbitrary $\mathcal{ELU}$ ontology into datalog unless PTIME = NP. Furthermore, Lutz and Wolter [2012] showed that tractability w.r.t. data complexity cannot be achieved for an arbitrary non-Horn ontology $\mathcal{T}$ with 'real' disjunctions: for each such $\mathcal{T}$, a query $Q$ exists such that answering $Q$ w.r.t. $\mathcal{T}$ is co-NP-hard.

The result by Lutz and Wolter [2012], however, depends on an interaction between existentially quantified variables in $Q$ and disjunctions in $\mathcal{T}$. Motivated by this observation, we consider the problem of computing datalog rewritings of *ground* queries (i.e., queries whose answers must map all the variables in $Q$ to constants) over non-Horn ontologies. Apart from allowing us to overcome the negative result by Lutz and Wolter [2012], this also allows us to compute a rewriting of $\mathcal{T}$ that can be used to answer an arbitrary ground query. Such queries form the basis of SPARQL, which makes our results practically relevant. We summarise our results as follows.

In Section 3, we revisit the limits of datalog rewritability for a language as a whole and show that non-rewritability of $\mathcal{ELU}$ ontologies is independent from any complexity-theoretic assumptions. More precisely, we present an $\mathcal{ELU}$ ontology $\mathcal{T}$ for which query answering cannot be decided by a family of monotone circuits of polynomial size, which contradicts the results by Afrati *et al.* [1995], who proved that

*Work supported by the Royal Society, the EPSRC projects Score!, Exoda, and MaSI[3], and an FP7 Marie Curie Fellowship.

fact entailment in a fixed datalog program can be decided using monotone circuits of polynomial size. Thus, instead of relying on complexity arguments, we compare the lengths of proofs in $\mathcal{ELU}$ and datalog and show that the proofs in $\mathcal{ELU}$ may be considerably longer than the proofs in datalog.

In Section 4, we present a three-step procedure that takes a $\mathcal{SHI}$-ontology $\mathcal{T}$ and attempts to rewrite $\mathcal{T}$ into a datalog program. First, we use a novel technique to rewrite $\mathcal{T}$ into a TBox $\Omega_{\mathcal{T}}$ without transitivity axioms while preserving entailment of *all* ground atoms; this is in contrast to the standard techniques (see, e.g., [Hustadt *et al.*, 2007]), which preserve entailments only of unary facts and binary facts with roles not having transitive subroles. Second, we use the algorithm by Hustadt *et al.* [2007] to rewrite $\Omega_{\mathcal{T}}$ into a *disjunctive datalog program* $\mathsf{DD}(\Omega_{\mathcal{T}})$. Third, we adapt the knowledge compilation technique by del Val [2005] and Selman and Kautz [1996] to transform $\mathsf{DD}(\Omega_{\mathcal{T}})$ into a datalog program. The final step is not guaranteed to terminate in general; however, if it terminates, the resulting program is a rewriting of $\mathcal{T}$.

In Section 4.4, we show that our procedure always terminates if $\mathcal{T}$ is a DL-Lite$^{\mathcal{H},+}_{bool}$-ontology—a practically-relevant language that extends OWL 2 QL with transitive roles and Boolean connectives. Artale *et al.* [2009] proved that the data complexity of concept queries in this language is tractable (i.e., NLogSpace-complete). We extend this result to all ground queries and thus obtain a goal-oriented rewriting algorithm that may be suitable for practical use.

Our technique, as well as most rewriting techniques known in the literature, is based on a sound inference system and thus produces only *strong rewritings*—that is, rewritings entailed by the original ontology. In Section 5 we show that non-Horn ontologies exist that can be rewritten into datalog, but that have no strong rewritings. This highlights the limits of techniques based on sound inferences. It is also surprising since all known rewriting techniques for Horn fragments of OWL 2 DL known to us produce only strong rewritings.

The proofs of all of our technical results are given in the accompanying technical report [Cuenca Grau *et al.*, 2013].

## 2 Preliminaries

We consider first-order logic without equality and function symbols. Variables, terms, (ground) atoms, literals, formulae, sentences, interpretations $I = (\Delta^I, \cdot^I)$, models, and entailment ($\models$) are defined as usual. We call a finite set of facts (i.e., ground atoms) an *ABox*. We write $\varphi(\vec{x})$ to stress that a first-order formula $\varphi$ has free variables $\vec{x} = x_1, \ldots, x_n$.

**Resolution Theorem Proving**

We use the standard notions of (Horn) clauses, substitutions (i.e., mappings of variables to terms), and most general unifiers (MGUs). We often identify a clause with the set of its literals. *Positive factoring* (PF) and *binary resolution* (BR) are as follows, where $\sigma$ is the MGU of atoms $A$ and $B$:

$$\text{PF:}\quad \frac{C \vee A \vee B}{C\sigma \vee A\sigma} \qquad \text{BR:}\quad \frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma}$$

A clause $C$ is a *tautology* if it contains literals $A$ and $\neg A$. A clause $C$ subsumes a clause $D$ if a substitution $\sigma$ exists such that each literal in $C\sigma$ occurs in $D$. Furthermore, $C$ $\theta$-subsumes $D$ if $C$ subsumes $D$ and $C$ has no more literals than $D$. Finally, $C$ is *redundant* in a set of clauses $\mathcal{S}$ if $C$ is a tautology or if $C$ is $\theta$-subsumed by another clause in $\mathcal{S}$.

**Datalog and Disjunctive Datalog**

A *disjunctive rule* $r$ is a function-free first-order sentence of the form $\forall\vec{x}\forall\vec{z}.[\varphi(\vec{x},\vec{z}) \rightarrow \psi(\vec{x})]$, where tuples of variables $\vec{x}$ and $\vec{z}$ are disjoint, $\varphi(\vec{x},\vec{z})$ is a conjunction of atoms, and $\psi(\vec{x})$ is a disjunction of atoms. Formula $\varphi$ is the *body* of $r$, and formula $\psi$ is the *head* of $r$. For brevity, we often omit the quantifiers in a rule. A *datalog rule* is a disjunctive rule where $\psi(\vec{x})$ is a single atom. A (disjunctive) datalog program $\mathcal{P}$ is a finite set of (disjunctive) datalog rules. Rules obviously correspond to clauses, so we sometimes abuse our definitions and use these two notions as synonyms. The *evaluation* of $\mathcal{P}$ over an ABox $\mathcal{A}$ is the set $\mathcal{P}(\mathcal{A})$ of facts entailed by $\mathcal{P} \cup \mathcal{A}$.

**Ontologies and Description Logics**

A DL *signature* is a disjoint union of sets of *atomic concepts*, *atomic roles*, and *individuals*. A *role* is an atomic role or an *inverse role* $R^-$ for $R$ an atomic role; furthermore, let $\mathsf{inv}(R) = R^-$ and $\mathsf{inv}(R^-) = R$. A *concept* is an expression of the form $\top$, $\bot$, $A$, $\neg C$, $C_1 \sqcap C_2$, $C_1 \sqcup C_2$, $\exists R.C$, $\forall R.C$, or $\exists R.\mathsf{self}$, where $A$ is an atomic concept, $C_{(i)}$ are concepts, and $R$ is a role. Concepts $\exists R.\mathsf{self}$ correspond to atoms $R(x,x)$ and are typically not included in $\mathcal{SHI}$; however, we use this minor extension in Section 4.1. A $\mathcal{SHI}$-TBox $\mathcal{T}$, often called an *ontology*, is a finite set of axioms of the form $R_1 \sqsubseteq R_2$ (*role inclusion axioms* or RIAs), $\mathsf{Tra}(R)$ (*transitivity axioms*), and $C_1 \sqsubseteq C_2$ (*general concept inclusions* or GCIs), where $R_{(i)}$ are roles and $C_{(i)}$ are concepts. Axiom $C_1 \equiv C_2$ abbreviates $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. Relation $\sqsubseteq^*_{\mathcal{T}}$ is the smallest reflexively–transitively closed relation such that $R \sqsubseteq^*_{\mathcal{T}} S$ and $\mathsf{inv}(R) \sqsubseteq^*_{\mathcal{T}} \mathsf{inv}(S)$ for each $R \sqsubseteq S \in \mathcal{T}$. A role $R$ is *transitive* in $\mathcal{T}$ if $\mathsf{Tra}(R) \in \mathcal{T}$ or $\mathsf{Tra}(\mathsf{inv}(R)) \in \mathcal{T}$. Satisfaction of a $\mathcal{SHI}$-TBox $\mathcal{T}$ in an interpretation $I = (\Delta^I, \cdot^I)$, written $I \models \mathcal{T}$, is defined as usual [Baader *et al.*, 2003].

An $\mathcal{ALCHI}$-TBox is a $\mathcal{SHI}$-TBox with no transitivity axioms. An $\mathcal{ELU}$-TBox is an $\mathcal{ALCHI}$-TBox with no role inclusion axioms, inverse roles, concepts $\exists R.\mathsf{self}$, or symbols $\bot$, $\forall$, and $\neg$. A DL-Lite$^{\mathcal{H},+}_{bool}$-TBox is a $\mathcal{SHI}$-TBox that does not contain concepts of the form $\forall R.C$, and where $C = \top$ for each concept of the form $\exists R.C$. The notion of *acyclic* TBoxes is defined as usual [Baader *et al.*, 2003].

A $\mathcal{SHI}$-TBox $\mathcal{T}$ is *normalised* if $\forall$ does not occur in $\mathcal{T}$, and $\exists$ occurs in $\mathcal{T}$ only in axioms of the form $\exists R.C \sqsubseteq A$, $\exists R.\mathsf{self} \sqsubseteq A$, $A \sqsubseteq \exists R.C$, or $A \sqsubseteq \exists R.\mathsf{self}$. Each $\mathcal{SHI}$-TBox $\mathcal{T}$ can be transformed in polynomial time into a normalised $\mathcal{SHI}$-TBox that is a model-conservative extension of $\mathcal{T}$.

**Queries and Datalog Rewritings**

A *ground query* (or just a *query*) $Q(\vec{x})$ is a conjunction of function-free atoms. A substitution $\sigma$ mapping $\vec{x}$ to constants is an *answer* to $Q(\vec{x})$ w.r.t. a set $\mathcal{F}$ of first-order sentences and an ABox $\mathcal{A}$ if $\mathcal{F} \cup \mathcal{A} \models Q(\vec{x})\sigma$; furthermore, $\mathsf{cert}(Q, \mathcal{F}, \mathcal{A})$ is the set of all answers to $Q(\vec{x})$ w.r.t. $\mathcal{F}$ and $\mathcal{A}$.

Let $Q$ be a query. A datalog program $\mathcal{P}$ is a *$Q$-rewriting* of a finite set of sentences $\mathcal{F}$ if $\mathsf{cert}(Q, \mathcal{F}, \mathcal{A}) = \mathsf{cert}(Q, \mathcal{P}, \mathcal{A})$ for each ABox $\mathcal{A}$. The program $\mathcal{P}$ is a *rewriting of* $\mathcal{F}$ if $\mathcal{P}$

is a $Q$-rewriting of $\mathcal{F}$ for each query $Q$. Such rewritings are *strong* if, in addition, we also have $\mathcal{F} \models \mathcal{P}$.

## 3 The Limits of Datalog Rewritability

Datalog programs can be evaluated over an ABox $\mathcal{A}$ in polynomial time in the size of $\mathcal{A}$; hence, a co-NP-hard property of $\mathcal{A}$ cannot be decided by evaluating a fixed datalog program over $\mathcal{A}$ unless PTIME = NP. Krisnadhi and Lutz [2007] showed that answering ground queries is co-NP-hard in data complexity even for acyclic TBoxes expressed in $\mathcal{ELU}$—the simplest non-Horn extension of the basic description logic $\mathcal{EL}$. Thus, under standard complexity-theoretic assumptions, an acyclic $\mathcal{ELU}$-TBox and a ground query $Q$ exist for which there is no $Q$-rewriting of $\mathcal{T}$. In this section, we show that this holds even if PTIME = NP.

**Theorem 1.** *An acyclic $\mathcal{ELU}$-TBox $\mathcal{T}$ and a ground CQ $Q$ exist such that $\mathcal{T}$ is not $Q$-rewritable.*

Our proof uses several notions from circuit complexity [Wegener, 1987], and results of this flavour compare the sizes of proofs in different formalisms; thus, our result essentially says that proofs in $\mathcal{ELU}$ can be significantly longer than proofs in datalog. Let $<$ be the ordering on Boolean values defined by $\mathsf{f} < \mathsf{t}$; then, a Boolean function $f$ with $n$ inputs is *monotone* if $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$ holds for all $n$-tuples of Boolean values $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ such that $x_i \leq y_i$ for each $1 \leq i \leq n$. A decision problem can be seen as a family of Boolean functions $\{f_n\}$, where $f_n$ decides membership of each $n$-bit input. If each function $f_n$ is monotone, then $f_n$ can be realised by a monotone Boolean circuit $C_n$ (i.e., a circuit with $n$ input gates where all internal gates are AND- or OR-gates with unrestricted fan-in); the size of $C_n$ is the number of its edges. The family of circuits $\{C_n\}$ corresponding to $\{f_n\}$ has polynomial size if a polynomial $p(x)$ exists such that the size of each $C_n$ is bounded by $p(n)$.

We recall how non-3-colorability of an undirected graph $G$ with $s$ vertices corresponds to monotone Boolean functions. The maximum number of edges in $G$ is $m(s) = s(s-1)/2$, so graph $G$ is encoded as a string $\vec{x}$ of $m(s)$ bits, where bit $x_{i,j}$, $1 \leq i < j \leq s$, is $\mathsf{t}$ if and only if $G$ contains an edge between vertices $i$ and $j$. The non-3-colorability problem can then be seen as a family of Boolean functions $\{f_{m(s)}\}$, where function $f_{m(s)}$ handles all graphs with $s$ vertices and it evaluates to $\mathsf{t}$ on an input $\vec{x}$ iff the graph corresponding to $\vec{x}$ is non-3-colourable. Functions $f_n$ such that $n \neq m(s)$ for all $s$ are irrelevant since no graph is encoded using that many bits.

We prove our claim using a result by Afrati *et al.* [1995]: if a decision problem cannot be solved using a family of monotone circuits of polynomial size, then the problem also cannot be solved by evaluating a fixed datalog program, regardless of the problem's complexity. We restate the result as follows.

**Theorem 2.** *[Adapted from Afrati* et al. *1995]*

1. *Let $\mathcal{P}$ be a fixed datalog program, and let $\alpha$ be a fixed fact. Then, for an ABox $\mathcal{A}$, deciding $\mathcal{P} \cup \mathcal{A} \models \alpha$ can be solved by monotone circuits of polynomial size.*
2. *The non-3-colorability problem cannot be solved by monotone circuits of polynomial size.*

Table 1: Example TBox $\mathcal{T}_{\mathsf{ex}}$

| | |
|---|---|
| $\gamma_1$ | $\mathsf{Student} \sqsubseteq \mathsf{GrSt} \sqcup \mathsf{UnGrSt}$ |
| $\gamma_2$ | $\mathsf{Course} \sqsubseteq \mathsf{GrCo} \sqcup \mathsf{UnGrCo}$ |
| $\gamma_3$ | $\mathsf{PhDSt} \sqsubseteq \exists \mathsf{takes}.\mathsf{PhDCo}$ |
| $\gamma_4$ | $\mathsf{PhDCo} \sqsubseteq \mathsf{GrCo}$ |
| $\gamma_5$ | $\exists \mathsf{takes}.\mathsf{GrCo} \sqsubseteq \mathsf{GrSt}$ |
| $\gamma_6$ | $\mathsf{UnGrSt} \sqcap \exists \mathsf{takes}.\mathsf{GrCo} \sqsubseteq \bot$ |

To prove Theorem 1, we present a TBox $\mathcal{T}$ and a ground CQ $Q$ that decide non-3-colorability of a graph encoded as an ABox. Next, we present a family of monotone Boolean functions $\{g_{n(u)}\}$ that decide answering $Q$ w.r.t. $\mathcal{T}$ an arbitrary ABox $\mathcal{A}$. Next, we show that a monotone circuit for arbitrary $f_{m(s)}$ can be obtained by a size-preserving transformation from a circuit for some $g_{n(u)}$; thus, by Item 2 of Theorem 2, answering $Q$ w.r.t. $\mathcal{T}$ cannot be solved using monotone circuits of polynomial size. Finally, we show that existence of a rewriting for $Q$ and $\mathcal{T}$ contradicts Item 1 of Theorem 2.

## 4 Computing Rewritings via Resolution

Theorem 1 is rather discouraging since it applies to one of the simplest non-Horn languages. The theorem's proof, however, relies on a specific TBox $\mathcal{T}$ that encodes a hard problem (i.e., non-3-colorability) that is not solvable by monotone circuits of polynomial size. One can expect that non-Horn TBoxes used in practice do not encode such hard problems, and so it might be possible to rewrite such TBoxes into datalog.

We illustrate this intuition using the TBox $\mathcal{T}_{\mathsf{ex}}$ shown in Table 1. Axioms $\gamma_4$–$\gamma_6$ correspond to datalog rules, whereas axioms $\gamma_1$–$\gamma_3$ represent disjunctive and existentially quantified knowledge and thus do not correspond to datalog rules. We will show that $\mathcal{T}_{\mathsf{ex}}$ can, in fact, be rewritten into datalog using a generic three-step method that takes a normalised $\mathcal{SHI}$-TBox $\mathcal{T}$ and proceeds as follows.

- **S1** Eliminate the transitivity axioms from $\mathcal{T}$ by transforming $\mathcal{T}$ into an $\mathcal{ALCHI}$-TBox $\Omega_{\mathcal{T}}$ and a set of datalog rules $\Xi_{\mathcal{T}}$ such that facts entailed by $\mathcal{T} \cup \mathcal{A}$ and $\Omega_{\mathcal{T}} \cup \Xi_{\mathcal{T}}(\mathcal{A})$ coincide for each ABox $\mathcal{A}$. This step extends the known technique to make it complete for facts with roles that have transitive subroles in $\mathcal{T}$.
- **S2** Apply the algorithm by Hustadt *et al.* [2007] to transform $\Omega_{\mathcal{T}}$ into a disjunctive datalog program $\mathsf{DD}(\Omega_{\mathcal{T}})$.
- **S3** Transform $\mathsf{DD}(\Omega_{\mathcal{T}})$ into a set of datalog rules $\mathcal{P}_H$ using a variant of the knowledge compilation techniques by Selman and Kautz [1996] and del Val [2005].

Step **S3** may not terminate for an arbitrary $\mathcal{SHI}$-TBox $\mathcal{T}$; however, if it terminates (i.e., if $\mathcal{P}_H$ is finite), then $\mathcal{P}_H \cup \Xi_{\mathcal{T}}$ is a rewriting of $\mathcal{T}$. Furthermore, in Section 4.4 we show that step **S3** always terminates if $\mathcal{T}$ is a DL-Lite$^{\mathcal{H},+}_{bool}$-TBox. We thus obtain what is, to the best of our knowledge, the first goal-oriented rewriting algorithm for a practically-relevant non-Horn fragment of OWL 2 DL.

### 4.1 Transitivity

We first recapitulate the standard technique for eliminating transitivity axioms from $\mathcal{SHI}$-TBoxes.

**Definition 3.** *Let $\mathcal{T}$ be a normalised $\mathcal{SHI}$-TBox, and let $\Theta_\mathcal{T}$ be obtained from $\mathcal{T}$ by removing all transitivity axioms. If $\mathcal{T}$ is a DL-Lite$^{\mathcal{H},+}_{bool}$-TBox, then let $\Upsilon_\mathcal{T} = \Theta_\mathcal{T}$; otherwise, let $\Upsilon_\mathcal{T}$ be the extension of $\Theta_\mathcal{T}$ with axioms*

$$\exists R.A \sqsubseteq C_{B,R} \qquad \exists R.C_{B,R} \sqsubseteq C_{B,R} \qquad C_{B,R} \sqsubseteq B$$

*for each axiom $\exists S.A \sqsubseteq B \in \mathcal{T}$ and each transitive role $R$ in $\mathcal{T}$ such that $R \sqsubseteq^*_\mathcal{T} S$, where $C_{B,R}$ is a fresh atomic concept unique for $B$ and $R$.*

This encoding preserves entailment of all facts of the form $C(c)$ and $U(c,d)$ if $U$ has no transitive subroles: this was proved by Artale *et al.* [2009] for DL-Lite$^{\mathcal{H},+}_{bool}$, and by Simancik [2012] for $\mathcal{SHI}$. Example 4, however, shows that the encoding is incomplete if $U$ has transitive subroles.

**Example 4.** *Let $\mathcal{T}$ be the TBox below, and let $\mathcal{A} = \{A(a)\}$.*

$$A \sqsubseteq \exists S.B \qquad S \sqsubseteq R \qquad S \sqsubseteq R^- \qquad \mathsf{Tra}(R)$$

*Then, $\Upsilon_\mathcal{T} = \mathcal{T} \setminus \{\mathsf{Tra}(R)\}$, and one can easily verify that $\mathcal{T} \cup \mathcal{A} \models R(a,a)$, but $\Upsilon_\mathcal{T} \cup \mathcal{A} \not\models R(a,a)$. Note, however, that the missing inference can be recovered by extending $\Upsilon_\mathcal{T}$ with the axiom $A \sqsubseteq \exists R.\mathsf{self}$, which is a consequence of $\mathcal{T}$.*

The intuitions from Example 4 are formalised in Definition 5. Roughly speaking, we transform the transitivity and role inclusion axioms in $\mathcal{T}$ into a datalog program $\Xi_\mathcal{T}$, which we apply to $\mathcal{A}$ 'first'—that is, we compute $\Xi_\mathcal{T}(\mathcal{A})$ independently from any GCIs. To recoup the remaining consequences of the form $R(a,a)$, we extend $\Upsilon_\mathcal{T}$ with sufficiently many axioms of the form $A \sqsubseteq \exists R.\mathsf{self}$ that are entailed by $\mathcal{T}$; this is possible since we assume that $\mathcal{T}$ is normalised.

**Definition 5.** *Let $\mathcal{T}$ be a normalised $\mathcal{SHI}$-TBox. Then, $\Omega_\mathcal{T}$ is the TBox obtained by extending $\Upsilon_\mathcal{T}$ with an axiom $A \sqsubseteq \exists R.\mathsf{self}$ for each atomic concept $A$ and each atomic role $R$ such that $R$ is transitive in $\mathcal{T}$, and $A \sqsubseteq \exists S.B \in \mathcal{T}$ for some concept $B$ and role $S$ with $S \sqsubseteq^*_\mathcal{T} R$ and $S \sqsubseteq^*_\mathcal{T} R^-$. Furthermore, $\Xi_\mathcal{T}$ is the set of datalog rules corresponding to the role inclusion and transitivity axioms in $\mathcal{T}$.*

**Theorem 6.** *Let $\mathcal{T}$ be a normalised $\mathcal{SHI}$-TBox, let $\mathcal{A}$ be an ABox, and let $\alpha$ be a fact. Then, $\mathcal{T} \cup \mathcal{A} \models \alpha$ if and only if $\Omega_\mathcal{T} \cup \Xi_\mathcal{T}(\mathcal{A}) \models \alpha$.*

Note that, if $\mathcal{T}$ is normalised, so is $\Omega_\mathcal{T}$. Furthermore, to ensure decidability, roles involving transitive subroles are not allowed occur in $\mathcal{T}$ in number restrictions, and so Theorem 6 holds even if $\mathcal{T}$ is a $\mathcal{SHOIQ}$-TBox.

## 4.2 From DLs to Disjunctive Datalog

Step **S2** of our rewriting algorithm uses the technique by Hustadt *et al.* [2007] for transforming an $\mathcal{ALCHI}$-TBox $\mathcal{T}$ into a disjunctive datalog program $\mathsf{DD}(\mathcal{T})$ such that, for each ABox $\mathcal{A}$, the facts entailed by $\mathcal{T} \cup \mathcal{A}$ and $\mathsf{DD}(\mathcal{T}) \cup \mathcal{A}$ coincide. By eliminating the existential quantifiers in $\mathcal{T}$, one thus reduces a reasoning problem in $\mathcal{T} \cup \mathcal{A}$ to a reasoning problem in $\mathsf{DD}(\mathcal{T}) \cup \mathcal{A}$. The following definition summarises the properties of the programs produced by the transformation.

**Definition 7.** *A disjunctive datalog program $\mathcal{P}$ is* nearly-monadic *if its rules can be partitioned into two disjoint sets, $\mathcal{P}^m$ and $\mathcal{P}^r$, such that*

Table 2: Example Disjunctive Program $\mathsf{DD}(\mathcal{T}_\mathsf{ex})$

| | |
|---|---|
| $C_1$ | $\neg\mathsf{Student}(x) \vee \mathsf{GrSt}(x) \vee \mathsf{UnGrSt}(x)$ |
| $C_2$ | $\neg\mathsf{Course}(x) \vee \mathsf{GrCo}(x) \vee \mathsf{UnGrCo}(x)$ |
| $C_3$ | $\neg\mathsf{PhDSt}(x) \vee \mathsf{GrSt}(x)$ |
| $C_4$ | $\neg\mathsf{PhDCo}(x) \vee \mathsf{GrCo}(x)$ |
| $C_5$ | $\neg\mathsf{takes}(x,y) \vee \neg\mathsf{GrCo}(y) \vee \mathsf{GrSt}(x)$ |
| $C_6$ | $\neg\mathsf{UnGrSt}(x) \vee \neg\mathsf{takes}(x,y) \vee \neg\mathsf{GrCo}(y)$ |

1. *each rule $r \in \mathcal{P}^m$ mentions only unary and binary predicates and each atom in the head of $r$ is of the form $A(z)$ or $R(z,z)$ for some variable $z$, and*
2. *each rule $r \in \mathcal{P}^r$ is of the form $R(x,y) \to S(x,y)$ or $R(x,y) \to S(y,x)$.*

*A disjunctive rule $r$ is* simple *if there exists a variable $x$ such that each atom in the body of $r$ is of the form $A_i(x)$, $R_i(x,x)$, $S_i(x,y_i)$, or $T_i(y_i,x)$, each atom in the head of $r$ is of the form $U_i(x,x)$ or $B_i(x)$, and each variable $y_i$ occurs in $r$ at most once. Furthermore, a nearly-monadic program $\mathcal{P}$ is* simple *if each rule in $\mathcal{P}^m$ is simple.*

Theorem 8 follows mainly from the results by Hustadt *et al.* [2007]; we just argue that concepts $\exists R.\mathsf{self}$ do not affect the algorithm, and that $\mathsf{DD}(\mathcal{T})$ satisfies property 1.

**Theorem 8.** *For $\mathcal{T}$ a normalised $\mathcal{ALCHI}$-TBox, $\mathsf{DD}(\mathcal{T})$ satisfies the following:*

1. *program $\mathsf{DD}(\mathcal{T})$ is nearly-monadic; furthermore, if $\mathcal{T}$ is a DL-Lite$^{\mathcal{H},+}_{bool}$-TBox, then $\mathsf{DD}(\mathcal{T})$ is also simple;*
2. *$\mathcal{T} \models \mathsf{DD}(\mathcal{T})$; and*
3. *$\mathsf{cert}(Q,\mathcal{T},\mathcal{A}) = \mathsf{cert}(Q,\mathsf{DD}(\mathcal{T}),\mathcal{A})$ for each ABox $\mathcal{A}$ and each ground query $Q$.*

**Example 9.** *When applied to the TBox $\mathcal{T}_\mathsf{ex}$ in Table 1, this algorithm produces the disjunctive program $\mathsf{DD}(\mathcal{T}_\mathsf{ex})$ shown (as clauses) in Table 2. In particular, axiom $\gamma_3$ is eliminated since it contains an existential quantifier, but its effects are compensated by clause $C_3$. Clauses $C_1$–$C_2$ and $C_4$–$C_6$ are obtained from axioms $\gamma_1$–$\gamma_2$ and $\gamma_4$–$\gamma_6$, respectively.*

## 4.3 From Disjunctive Datalog to Datalog

Step **S3** of our rewriting algorithm attempts to transform the disjunctive program obtained in Step **S2** into a datalog program such that, for each ABox $\mathcal{A}$, the two programs entail the same facts. This is achieved using known knowledge compilation techniques, which we survey next.

**Resolution-Based Knowledge Compilation**

In their seminal paper, Selman and Kautz [1996] proposed an algorithm for compiling a set of propositional clauses $\mathcal{S}$ into a set of Horn clauses $\mathcal{S}_H$ such that the Horn consequences of $\mathcal{S}$ and $\mathcal{S}_H$ coincide. Subsequently, del Val [2005] generalised this algorithm to the case when $\mathcal{S}$ contains first-order clauses, but without any termination guarantees; Procedure 1 paraphrases this algorithm. The algorithm applies to $\mathcal{S}$ binary resolution and positive factoring from resolution theorem proving, and it keeps only the consequences that are not redundant according to Definition 10. Unlike standard resolution, the algorithm maintains two sets $\mathcal{S}_H$ and $\mathcal{S}_{\overline{H}}$ of Horn

**Procedure 1** Compile-Horn

**Input:** $\mathcal{S}$: set of clauses
**Output:** $\mathcal{S}_H$: set of Horn clauses

1: $\mathcal{S}_H := \{C \in \mathcal{S} \mid C \text{ is a Horn clause and not a tautology}\}$
2: $\mathcal{S}_{\overline{H}} := \{C \in \mathcal{S} \mid C \text{ is a non-Horn clause and not a tautology}\}$
3: **repeat**
4: Compute all relevant consequences of $\langle \mathcal{S}_H, \mathcal{S}_{\overline{H}} \rangle$
5: **for each** relevant consequence $C$ of $\langle \mathcal{S}_H, \mathcal{S}_{\overline{H}} \rangle$ **do**
6: Delete from $\mathcal{S}_H$ and $\mathcal{S}_{\overline{H}}$ all clauses $\theta$-subsumed by $C$
7: **if** $C$ is Horn **then** $\mathcal{S}_H := \mathcal{S}_H \cup \{C\}$
8: **else** $\mathcal{S}_{\overline{H}} := \mathcal{S}_{\overline{H}} \cup \{C\}$
9: **until** there is no relevant consequence of $\langle \mathcal{S}_H, \mathcal{S}_{\overline{H}} \rangle$
10: **return** $\mathcal{S}_H$

and non-Horn clauses, respectively; furthermore, the algorithm never resolves two Horn clauses.

**Definition 10.** *Let $\mathcal{S}_H$ and $\mathcal{S}_{\overline{H}}$ be sets of Horn and non-Horn clauses, respectively. A clause $C$ is a* relevant consequence *of $\langle \mathcal{S}_H, \mathcal{S}_{\overline{H}} \rangle$ if*

- *$C$ is not redundant in $\mathcal{S}_H \cup \mathcal{S}_{\overline{H}}$, and*
- *$C$ is a factor of a clause $C_1 \in \mathcal{S}_{\overline{H}}$, or a resolvent of clauses $C_1 \in \mathcal{S}_{\overline{H}}$ and $C_2 \in \mathcal{S}_{\overline{H}} \cup \mathcal{S}_H$.*

Theorem 11 recapitulates the algorithm's properties. It essentially shows that, even if the algorithm never terminates, each Horn consequence of $\mathcal{S}$ will at some point during algorithm's execution become entailed by the set of Horn clauses $\mathcal{S}_H$ computed by the algorithm. The theorem was proved by showing that each resolution proof of a consequence of $\mathcal{S}$ can be transformed to 'postpone' all resolution steps between two Horn clauses until the end; thus, one can 'precompute' set $\mathcal{S}_H$ of all consequences of $\mathcal{S}$ derivable using a non-Horn clause.

**Theorem 11.** *([del Val, 2005]) Let $\mathcal{S}$ be a set of clauses, and let $C$ be a Horn clause such that $\mathcal{S} \models C$, and assume that Procedure 1 is applied to $\mathcal{S}$. Then, after some finite number of iterations of the loop in lines 3–9, we have $\mathcal{S}_H \models C$.*

**ABox-Independent Compilation**

Compiling knowledge into Horn clauses and computing datalog rewritings are similar in spirit: both transform one theory into another while ensuring that the two theories are indistinguishable w.r.t. a certain class of queries. There is, however, an important difference: given a disjunctive program $\mathcal{P}$ and a fixed ABox $\mathcal{A}$, one could apply Procedure 1 to $\mathcal{S} = \mathcal{P} \cup \mathcal{A}$ to obtain a datalog program $\mathcal{S}_H$, but such $\mathcal{S}_H$ would not necessarily be independent from the specific ABox $\mathcal{A}$. In contrast, a rewriting of $\mathcal{P}$ is a datalog program $\mathcal{P}_H$ that can be freely combined with an arbitrary ABox $\mathcal{A}$. We next show that a program $\mathcal{P}_H$ satisfying the latter requirement can be obtained by applying Procedure 1 to $\mathcal{P}$ only.

Towards this goal, we generalise Theorem 11 and show that, when applied to an arbitrary set of first-order clauses $\mathcal{N}$, Procedure 1 computes a set of Horn clauses $\mathcal{N}_H$ such that the Horn consequences of $\mathcal{N} \cup \mathcal{A}$ and $\mathcal{N}_H \cup \mathcal{A}$ coincide for an arbitrary ABox $\mathcal{A}$. Intuitively, this shows that, when Procedure 1 is applied to $\mathcal{S} = \mathcal{N} \cup \mathcal{A}$, all inferences involving facts in $\mathcal{A}$ can be 'moved' to end of derivations.

**Theorem 12.** *Let $\mathcal{N}$ be a set of clauses, let $\mathcal{A}$ be an ABox, let $C$ be a Horn clause such that $\mathcal{N} \cup \mathcal{A} \models C$, and assume that Procedure 1 is applied to $\mathcal{N}$. Then, after some finite number of iterations of the loop in lines 3–9, we have $\mathcal{N}_H \cup \mathcal{A} \models C$.*

**Rewriting Nearly-Monadic Disjunctive Programs**

The final obstacle to obtaining a datalog rewriting of a $\mathcal{SHI}$-TBox $\mathcal{T}$ is due to Theorem 6: the rules in $\Xi_\mathcal{T}$ should be applied 'before' $\Omega_\mathcal{T}$. While this allows us to transform $\Omega_\mathcal{T}$ into $\mathcal{P} = \mathsf{DD}(\Omega_\mathcal{T})$ and $\mathcal{P}_H$ without taking $\Xi_\mathcal{T}$ into account, this also means that Theorems 6, 8, and 12 only imply that the facts entailed by $\mathcal{T} \cup \mathcal{A}$ and $\mathcal{P}_H \cup \Xi_\mathcal{T}(\mathcal{A})$ coincide. To obtain a 'true' rewriting, we show in Lemma 13 that program $\mathcal{P}_H$ is nearly-monadic. We use this observation in Theorem 14 to show that each binary fact obtained by applying $\mathcal{P}_H$ to $\Xi_\mathcal{T}(\mathcal{A})$ is of the form $R(c, c)$, and so it cannot 'fire' the rules in $\Xi_\mathcal{T}$; hence, $\mathcal{P}_H \cup \Xi_\mathcal{T}$ is a rewriting of $\mathcal{T}$.

**Lemma 13.** *Let $\mathcal{P}$ be a nearly-monadic program, and assume that Procedure 1 terminates when applied to $\mathcal{P}$ and returns $\mathcal{P}_H$. Then, $\mathcal{P}_H$ is a nearly-monadic datalog program.*

**Theorem 14.** *Let $\mathcal{P} = \mathsf{DD}(\Omega_\mathcal{T})$ for $\mathcal{T}$ an $\mathcal{SHI}$-TBox. If, when applied to $\mathcal{P}$, Procedure 1 terminates and returns $\mathcal{P}_H$, then $\mathcal{P}_H \cup \Xi_\mathcal{T}$ is a rewriting of $\mathcal{T}$.*

Please note that our algorithm (just like all rewriting algorithms we are aware of) computes rewritings using a sound inference system and thus always produces strong rewritings.

**Example 15.** *When applied to the program $\mathcal{P} = \mathsf{DD}(\mathcal{T}_\mathsf{ex})$ from Table 2, Procedure 1 resolves $C_2$ and $C_5$ to derive* (1)*, $C_2$ and $C_6$ to derive* (2)*, and $C_1$ and $C_6$ to derive* (3)*.*

$$\neg\mathsf{takes}(x, y) \vee \neg\mathsf{Course}(y) \vee \mathsf{GrSt}(x) \vee \mathsf{UnGrCo}(y) \quad (1)$$
$$\neg\mathsf{takes}(x, y) \vee \neg\mathsf{UnGrSt}(x) \vee \neg\mathsf{Course}(y) \vee \mathsf{UnGrCo}(y) \quad (2)$$
$$\neg\mathsf{takes}(x, y) \vee \neg\mathsf{Student}(x) \vee \neg\mathsf{GrCo}(y) \vee \mathsf{GrSt}(x) \quad (3)$$

*Resolving* (2) *and* $C_1$*, and* (3) *and* $C_2$ *produces redundant clauses, after which the procedure terminates and returns the set $\mathcal{P}_H$ consisting of clauses $C_3$–$C_6$,* (2)*, and* (3)*. By Theorem 14, $\mathcal{P}_H$ is a strong rewriting of $\mathcal{T}_\mathsf{ex}$.*

## 4.4 Termination

Procedure 1 is not a semi-decision procedure for either strong non-rewritability (cf. Example 16) or strong rewritability (cf. Example 17) of nearly-monadic programs.

**Example 16.** *Let $\mathcal{P}$ be defined as follows.*

$$G(x) \vee B(x) \quad (4)$$
$$B(x_1) \vee \neg E(x_1, x_0) \vee \neg G(x_0) \quad (5)$$
$$G(x_1) \vee \neg E(x_1, x_0) \vee \neg B(x_0) \quad (6)$$

*Clauses* (5) *and* (6) *are mutually recursive, but they are also Horn, so Procedure 1 never resolves them directly.*

*Clauses* (5) *and* (6)*, however, can interact through clause* (4)*. Resolving* (4) *and* (5) *on $\neg G(x_0)$ produces* (7)*; and resolving* (6) *and* (7) *on $B(x_1)$ produces* (8)*. By further resolving* (8) *alternatively with* (5) *and* (6)*, we obtain* (9) *for each even $n$. By resolving* (6) *and* (9) *on $B(x_0)$, we obtain* (10)*. Finally, by factoring* (10)*, we obtain* (11) *for each even $n$.*

$$B(x_1) \vee \neg E(x_1, x_0) \vee B(x_0) \quad (7)$$

$$G(x_2) \vee \neg E(x_2, x_1) \vee \neg E(x_1, x_0) \vee B(x_0) \quad (8)$$

$$G(x_n) \vee [\bigvee_{i=1}^{n} \neg E(x_i, x_{i-1})] \vee B(x_0) \quad (9)$$

$$G(x_n) \vee [\bigvee_{i=1}^{n} \neg E(x_i, x_{i-1})] \vee G(x'_1) \vee \neg E(x'_1, x_0) \quad (10)$$

$$G(x_n) \vee \neg E(x_n, x_0) \vee [\bigvee_{i=1}^{n} \neg E(x_i, x_{i-1})] \quad (11)$$

*Procedure 1 thus derives on $\mathcal{P}$ an infinite set of Horn clauses, and Theorem 22 shows that no strong rewriting of $\mathcal{P}$ exists.*

**Example 17.** *Let $\mathcal{P}$ be defined as follows.*

$$B_1(x_0) \vee B_2(x_0) \vee \neg A(x_0) \quad (12)$$

$$A(x_1) \vee \neg E(x_1, x_0) \vee \neg B_1(x_0) \quad (13)$$

$$A(x_1) \vee \neg E(x_1, x_0) \vee \neg B_2(x_0) \quad (14)$$

*When applied to $\mathcal{P}$, Procedure 1 will eventually compute infinitely many clauses $C_n$ of the following form:*

$$C_n = A(x_n) \vee [\bigvee_{i=1}^{n} \neg E(x_i, x_{i-1})] \vee \neg A(x_0)$$

*However, for each $n > 1$, clause $C_n$ is a logical consequence of clause $C_1$, so the program consisting of clauses* (12)*,* (13)*, and $C_1$ is a strong rewriting of $\mathcal{P}$.*

Example 18 demonstrates another problem that can arise even if $\mathcal{P}$ is nearly-monadic and simple.

**Example 18.** *Let $\mathcal{P}$ be the following program:*

$$\neg R(x, y) \vee A(x) \quad (15)$$

$$\neg R(x, y) \vee B(x) \quad (16)$$

$$\neg A(x) \vee \neg B(x) \vee C(x) \vee D(x) \quad (17)$$

*Now resolving* (15) *and* (17) *produces* (18)*; and resolving* (16) *and* (18) *produces* (19)*.*

$$\neg R(x, y) \vee \neg B(x) \vee C(x) \vee D(x) \quad (18)$$

$$\neg R(x, y_1) \vee \neg R(x, y_2) \vee C(x) \vee D(x) \quad (19)$$

*Clause* (19) *contains more variables than clauses* (15) *and* (16)*, which makes bounding the clause size difficult.*

Notwithstanding Example 18, we believe one can prove that Procedure 1 terminates if $\mathcal{P}$ is nearly-monadic and simple. However, apart from making the termination proof more involved, deriving clauses such as (19) is clearly inefficient. We therefore extend Procedure 1 with the condensation simplification rule, which eliminates redundant literals in clauses such as (19). A *condensation* of a clause $C$ is a clause $D$ with the least number of literals such that $D \subseteq C$ and $C$ subsumes $D$. A condensation of $C$ is unique up to variable renaming, so we usually speak of *the* condensation of $C$. We next show that Theorems 11 and 12 hold even with condensation.

**Lemma 19.** *Theorems 11 and 12 hold if Procedure 1 is modified so that, after line 5, $C$ is replaced with its condensation.*

One can prove that all relevant consequences of nearly-monadic and simple clauses are also nearly-monadic and simple, so by using condensation to remove redundant literals, we obtain Lemma 20, which clearly implies Theorem 21.

**Lemma 20.** *If used with condensation, Procedure 1 terminates when applied to a simple nearly-monadic program $\mathcal{P}$.*

**Theorem 21.** *Let $\mathcal{P} = \mathsf{DD}(\Omega_\mathcal{T})$ for $\mathcal{T}$ a DL-Lite$^{\mathcal{H},+}_{bool}$-TBox. Procedure 1 with condensation terminates when applied to $\mathcal{P}$ and returns $\mathcal{P}_H$; furthermore, $\mathcal{P}_H \cup \Xi_\mathcal{T}$ is a rewriting of $\mathcal{T}$.*

We thus obtain a tractable (w.r.t. data complexity) procedure for answering queries over DL-Lite$^{\mathcal{H},+}_{bool}$-TBoxes. Furthermore, given a ground query $Q$ and a nearly-monadic and simple program $\mathcal{P}_H$ obtained by Theorem 21, it should be possible to match the NLogSpace lower complexity bound by Artale *et al.* [2009] as follows. First, one should apply backward chaining to $Q$ and $\mathcal{P}_H$ to compute a UCQ $Q'$ such that $\mathsf{cert}(Q, \mathcal{P}_H, \Xi_\mathcal{T}(\mathcal{A})) = \mathsf{cert}(Q', \emptyset, \Xi_\mathcal{T}(\mathcal{A}))$; since all nearly-monadic rules in $\mathcal{P}_H$ are simple, it should be possible to show that such 'unfolding' always terminates. Second, one should transform $\Xi_\mathcal{T}$ into an equivalent piecewise-linear datalog program $\Xi'_\mathcal{T}$. Although these transformations should be relatively straightforward, a formal proof would require additional machinery and is thus left for future work.

## 5 Limits to Strong Rewritability

We next show that strong rewritings may not exist for rather simple non-Horn $\mathcal{ELU}$-TBoxes that are rewritable in general. This is interesting because it shows that an algorithm capable of rewriting a larger class of TBoxes necessarily must depart from the common approaches based on sound inferences.

**Theorem 22.** *The $\mathcal{ELU}$-TBox $\mathcal{T}$ corresponding to the program $\mathcal{P}$ from Example 16 and the ground CQ $Q = G(x_1)$ are $Q$-rewritable, but not strongly $Q$-rewritable.*

The proof of Theorem 22 proceeds as follows. First, we show that, for each ABox $\mathcal{A}$ encoding a directed graph, we have $\mathsf{cert}(Q, \mathcal{T}, \mathcal{A}) \neq \emptyset$ iff the graph contains a pair of vertices reachable by both an even and an odd number of edges. Second, we show that latter property can be decided using a datalog program that uses new relations not occurring in $\mathcal{T}$. Third, we construct an infinite set of rules $\mathcal{R}$ entailed by each strong rewriting of $\mathcal{T}$. Fourth, we show that $\mathcal{R}' \not\models \mathcal{R}$ holds for each finite datalog program $\mathcal{R}'$ such that $\mathcal{T} \models \mathcal{R}'$.

Since our procedure from Section 4 produces only strong rewritings, it cannot terminate on a TBox that has no strong rewritings. This is illustrated in Example 16, which shows that Procedure 1 does not terminate when applied to (the clausification of) the TBox from Theorem 22.

## 6 Outlook

Our work opens many possibilities for future research. On the theoretical side, we will investigate whether one can decide existence of a strong rewriting for a given $\mathcal{SHI}$-TBox $\mathcal{T}$, and to modify Procedure 1 so that termination is guaranteed. Bienvenue *et al.* [2013] recently showed that rewritability of unary ground queries over $\mathcal{ALC}$-TBoxes is decidable; however, their result does not consider strong rewritability or binary ground queries. On the practical side, we will investigate whether Procedure 1 can be modified to use ordered resolution instead of unrestricted resolution. We will also implement our technique and evaluate its applicability.

## References


[Acciarri *et al.*, 2005] Andrea Acciarri, Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati. Quonto: Querying ontologies. In *AAAI*, pages 1670–1671, 2005.

[Afrati *et al.*, 1995] Foto Afrati, Stavros S Cosmadakis, and Mihalis Yannakakis. On Datalog vs. polynomial time. *J. of Computer and System Sciences*, 51(2), 1995.

[Artale *et al.*, 2009] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-Lite family and relations. *Journal of Artificial Intelligence Research*, 36:1–69, 2009.

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge Univ. Press, 2003.

[Baader *et al.*, 2005] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the el envelope. In *IJCAI*, pages 364–369, 2005.

[Bienvenue *et al.*, 2013] M. Bienvenue, B. Ten Cate, C. Lutz, and F. Wolter. Ontology-based Data Access: A Study through Disjunctive Datalog, CSP, and MMSNP. In *ACM PODS*, 2013.

[Bishop *et al.*, 2011] Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. OWLim: A family of scalable semantic repositories. *Semantic Web J.*, 2(1):33–42, 2011.

[Calvanese *et al.*, 2007] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Automated Reasoning (JAR)*, 39(3):385–429, 2007.

[Chortaras *et al.*, 2011] A. Chortaras, D. Trivela, and G. Stamou. Optimized query rewriting in OWL 2 QL. In *CADE*, pages 192–206, 2011.

[Cuenca Grau *et al.*, 2013] Bernardo Cuenca Grau, Boris Motik, Giorgos Stoilos, and Ian Horrocks. Computing Datalog Rewritings beyond Horn Ontologies, 2013. arXiv:1304.1402.

[del Val, 2005] Alvaro del Val. First order lub approximations: characterization and algorithms. *Artif. Intell.*, 162(1-2):7–48, 2005.

[Eiter *et al.*, 2012] Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for horn-shiq plus rules. In *AAAI*, 2012.

[Glimm *et al.*, 2008] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive Query Answering for the Description Logic $\mathcal{SHIQ}$. *Journal of Artificial Intelligence Research*, 31:151–198, 2008.

[Grosof *et al.*, 2003] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *WWW*, pages 48–57, 2003.

[Hustadt *et al.*, 2005] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Data complexity of reasoning in very expressive description logics. In *IJCAI*, pages 466–471, 2005.

[Hustadt *et al.*, 2007] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.

[Krisnadhi and Lutz, 2007] Adila Krisnadhi and Carsten Lutz. Data complexity in the $\mathcal{EL}$ family of description logics. In *LPAR*, 2007.

[Lutz and Wolter, 2012] Carsten Lutz and Frank Wolter. Non-Uniform Data Complexity of Query Answering in Description Logics. In *KR*, 2012.

[Lutz, 2008] C. Lutz. The Complexity of Conjunctive Query Answering in Expressive Description Logics. In *IJCAR*, 2008.

[Ortiz *et al.*, 2011] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Query answering in the horn fragments of the description logics shoiq and sroiq. In *IJCAI*, pages 1039–1044, 2011.

[Pérez-Urbina *et al.*, 2010] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic (JAL)*, 8(2):186–209, 2010.

[Rodriguez-Muro and Calvanese, 2012] Mariano Rodriguez-Muro and Diego Calvanese. High performance query answering over DL-Lite ontologies. In *KR*, 2012.

[Rosati and Almatelli, 2010] Riccardo Rosati and Alessandro Almatelli. Improving query answering over dl-lite ontologies. In *KR*, 2010.

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

[Simancik, 2012] F. Simancik. Elimination of Complex RIAs without Automata. In Y. Kazakov, D. Lembo, and F. Wolter, editors, *Proc. of the 2012. Int. Workshop on Description Logics (DL 2012)*, volume 846 of *CEUR Workshop Proceedings*, Rome, Italy, June 7–10 2012.

[Stocker and Smith, 2008] Markus Stocker and Michael Smith. Owlgres: A scalable owl reasoner. In *OWLED*, 2008.

[Wegener, 1987] I. Wegener. *The Complexity of Boolean Functions*. John Wiley and Sons, 1987.

[Wu *et al.*, 2008] Zhe Wu, George Eadon, Souripriya Das, Eugene Inseok Chong, Vladimir Kolovski, Melliyal Annamalai, and Jagannathan Srinivasan. Implementing an inference engine for RDFS/OWL constructs and user-defined rules in Oracle. In *ICDE*, pages 1239–1248, 2008.