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 appendices A–F.

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 φ 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 σ 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 σ exists such that each literal in $C\sigma$ occurs in D. Furthermore, C θ-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 θ-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 φ is the *body* of r, and formula ψ 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 σ 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 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 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 α 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 $\mathcal{T}_{\mathsf{ex}}$

γ_1	$\mathsf{Student} \sqsubseteq \mathsf{GrSt} \sqcup \mathsf{UnGrSt}$
γ_2	$\mathsf{Course} \sqsubseteq \mathsf{GrCo} \sqcup \mathsf{UnGrCo}$
γ_3	$\mathsf{PhDSt} \sqsubseteq \exists \mathsf{takes}.\mathsf{PhDCo}$
γ_4	$\mathsf{PhDCo} \sqsubseteq \mathsf{GrCo}$
γ_5	$\exists \mathsf{takes}.\mathsf{GrCo} \sqsubseteq \mathsf{GrSt}$
γ_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 γ_4–γ_6 correspond to datalog rules, whereas axioms γ_1–γ_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 α 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 γ_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 γ_1–γ_2 and γ_4–γ_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 θ-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.*, 2012] Bernardo Cuenca Grau, Boris Motik, Giorgos Stoilos, and Ian Horrocks. Completeness Guarantees for Incomplete Ontology Reasoners: Theory and Practice. *Journal of Artificial Intelligence Research*, 43:419–476, 2012.

[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.

[Motik *et al.*, 2009] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.

[Ortiz *et al.*, 2011] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Query answering in the horn fragments of the description logics $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$. 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.

A Proofs for Section 3

Before presenting the proof of Theorem 1, we recapitulate the definition of *monotone polynomial projections*, which are frequently used to transfer bounds on the circuit size from one family of monotone Boolean functions to another. Let f be a monotone Boolean function with inputs $\vec{x}$, and let g be a monotone Boolean function with inputs $\vec{y}$. Then, f is a *monotone projection* of g if a mapping $\rho : \vec{y} \to \{\mathsf{f}, \mathsf{t}\} \cup \vec{x}$ exists such that $f(\vec{x}) = g(\rho(\vec{y}))$ for each value of $\vec{x}$. Given such a mapping ρ, a monotone circuit that computes $g(\vec{y})$ can be transformed to a monotone circuit that computes $f(\vec{x})$ by replacing each input $y_i \in \vec{y}$ with $\rho(y_i)$. Furthermore, a family of Boolean functions $\{f_n\}$ is a *polynomial monotone projection* of a family $\{g_k\}$ if a polynomial $p(n)$ exists such that each f_n is a monotone projection of some g_k with $k \leq p(n)$; if that is the case and the family of functions $\{g_k\}$ can be realised by a family of monotone circuits of polynomial size, then so can $\{f_n\}$.

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

Proof. Let $\mathcal{T}$ be the following acyclic $\mathcal{ELU}$-TBox:

$$\begin{array}{rlrl} F_R &\equiv R \sqcap \exists \text{edge}.R & F_B &\equiv B \sqcap \exists \text{edge}.B \\ F_G &\equiv G \sqcap \exists \text{edge}.G & F &\equiv F_R \sqcup F_B \sqcup F_G \\ V &\sqsubseteq R \sqcup G \sqcup B & NC &\equiv \exists \text{vertex}.F \end{array}$$

Furthermore, let v be a fixed individual, and let $Q = NC(v)$. We next represent the problem of answering Q over $\mathcal{T}$ and an arbitrary input ABox $\mathcal{A}$ using a family of monotone functions $\{g_{n(u)}\}$. The *input size* of $\mathcal{A}$ is the number u of individuals occurring in $\mathcal{A}$ different from the fixed individual v; we assume that these individuals are labelled $a_1, \ldots, a_u$. Furthermore, to unify the notation, let $a_{u+1} = v$. Using the signature of $\mathcal{T}$, one can then construct at most $n(u) = 2(u+1)^2 + 9(u+1)$ assertions; hence, we encode $\mathcal{A}$ using $n(u)$ bits $y^{\text{edge}}_{i,j}$, $y^{\text{vertex}}_{i,j}$, and y^A_i as follows:

- for each $R \in \{\text{edge}, \text{vertex}\}$ and $1 \leq i, j \leq u$, bit $y^R_{i,j}$ is t if and only if $R(a_i, a_j) \in \mathcal{A}$; and
- for each $A \in \{R, G, B, F_R, F_B, F_G, F, V, NC\}$, bit y^A_i is t if and only if $A(a_i) \in \mathcal{A}$.

The family of Boolean functions $\{g_{n(u)}\}$ is defined such that, given a vector of bits $\vec{y}$ encoding an ABox $\mathcal{A}$ of input size u, we have $g_{n(u)}(\vec{y}) = \mathsf{t}$ if and only if $\mathcal{T} \cup \mathcal{A} \models Q$. Since first-order logic is monotonic, each $g_{n(u)}$ is clearly monotone.

Let $\{f_{m(s)}\}$ be the family of monotone Boolean functions associated with non-3-colorability as defined in Section 3. We next show that $\{f_{m(s)}\}$ is a monotone polynomial projection of $\{g_{n(u)}\}$. To this end, we first show that, for each positive integer s, function $f_{m(s)}$ is a monotone projection of $g_{n(s)}$. Let ρ be the following mapping, where A is a placeholder for each concept from the signature of $\mathcal{T}$ different from V:

$$\rho(y^{\text{edge}}_{i,j}) = \rho(y^{\text{edge}}_{j,i}) = \begin{cases} x_{i,j} & \text{for } 1 \leq i < j \leq s \\ \mathsf{f} & \text{otherwise} \end{cases}$$

$$\rho(y^{\text{vertex}}_{i,j}) = \begin{cases} \mathsf{t} & \text{for } i = s+1 \text{ and } 1 \leq j \leq s \\ \mathsf{f} & \text{otherwise} \end{cases}$$

$$\rho(y_i)^V = \begin{cases} \mathsf{t} & \text{for } 1 \leq i \leq s \\ \mathsf{f} & \text{for } i = s+1 \end{cases}$$

$$\rho(y_i)^A = \mathsf{f} \quad \text{for } 1 \leq i \leq s+1$$

We now show that $f_{m(s)}(\vec{x}) = g_{n(s)}(\rho(\vec{y}))$ for each vector $\vec{x}$ of $m(s)$ bits. To this end, let G be the undirected graph associated with $\vec{x}$ containing nodes $1, \ldots, s$. It is straightforward to check that $\rho(\vec{y})$ is then a vector of $n(s)$ bits that encodes the ABox $\mathcal{A}_G$ with individuals $a_1, \ldots, a_s, a_{s+1} = v$ containing the following assertions:

- $\text{edge}(a_i, a_j)$ and $\text{edge}(a_j, a_i)$ for all $1 \leq i < j \leq s$ such that G contains an edge between i and j, and
- assertions $V(a_i)$ and $\text{vertex}(v, a_i)$ for each $1 \leq i \leq s$.

Furthermore, it is routine to check that G is non-3-colorable iff $\mathcal{T} \cup \mathcal{A}_G \models Q$; but then, by the definition of $f_{m(s)}$ and $g_{n(s)}$, we have $f_{m(s)}(\vec{x}) = g_{n(s)}(\rho(\vec{y}))$, as required. Finally, for $p(z) = z^2$, we clearly have $n(s) \leq (m(s))^2$. Thus, the family of monotone functions $\{f_{m(s)}\}$ is a monotone polynomial projection of the family of monotone functions $\{g_{n(s)}\}$.

The above observation, Item 2 of Theorem 2, and the properties of monotone polynomial projections imply that the query answering problem for Q and $\mathcal{T}$ cannot be solved using monotone circuits of polynomial size. Now assume that a datalog program $\mathcal{P}$ exists that is a Q-rewriting of $\mathcal{T}$. By Item 1 of Theorem 2, answering Q over $\mathcal{P}$, and so the problem of answering Q over $\mathcal{T}$ as well, can be solved using monotone circuits of polynomial size, which is a contradiction. □

B Proof of Theorem 6

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

Proof. We prove the contrapositive: for each fact α, we have $\mathcal{T} \cup \mathcal{A} \not\models \alpha$ if and only if $\Omega_{\mathcal{T}} \cup \Xi_{\mathcal{T}}(\mathcal{A}) \not\models \alpha$.

($\Rightarrow$) It is routine to show that $\Upsilon_{\mathcal{T}}$ is a model-conservative extension of $\mathcal{T}$ [Simancik, 2012]. Furthermore, for all concepts A and B, each atomic role R, and each role S such that $A \sqsubseteq \exists S.B \in \mathcal{T}$, $S \sqsubseteq^*_{\mathcal{T}} R$, and $S \sqsubseteq^*_{\mathcal{T}} R^-$, we clearly have $\mathcal{T} \models A \sqsubseteq \exists R.\mathsf{self}$. By these two properties, $\Omega_{\mathcal{T}}$ is a model-conservative extension of $\mathcal{T}$. Finally, it is obvious that $\mathcal{T} \models \Xi_{\mathcal{T}}$. Now consider an arbitrary fact α such that $\mathcal{T} \cup \mathcal{A} \not\models \alpha$. Then, an interpretation I exists such that $I \models \mathcal{T} \cup \mathcal{A}$ and $I \not\models \alpha$. Since $\Omega_{\mathcal{T}}$ is a model-conservative extension of $\mathcal{T}$ and $\mathcal{T} \models \Xi_{\mathcal{T}}$, an interpretation J exists such that $J \models \Omega_{\mathcal{T}}$ and $J \models \Xi_{\mathcal{T}}(\mathcal{A})$; furthermore, since α does not use the symbols occurring in $\Omega_{\mathcal{T}}$ but not in $\mathcal{T}$, we also have $J \not\models \alpha$. Thus, we have $\Omega_{\mathcal{T}} \cup \Xi_{\mathcal{T}}(\mathcal{A}) \not\models \alpha$, as required.

($\Leftarrow$) Consider an arbitrary fact α such that $\Omega_{\mathcal{T}} \cup \Xi_{\mathcal{T}}(\mathcal{A}) \not\models \alpha$. Then, an interpretation $I = (\Delta^I, \cdot^I)$ exists such that $I \models \Omega_{\mathcal{T}} \cup \Xi_{\mathcal{T}}(\mathcal{A})$ and $I \not\models \alpha$. Without loss of generality, we can assume that I is of a special tree shape, which we describe next. Let $N_{\mathcal{A}}$ be the set of individuals occurring in $\mathcal{A}$, and let N be the smallest set such that $N_{\mathcal{A}} \subseteq N$ and, if $u \in N$, then $u.i \in N$ for each nonnegative integer i. Then, we can assume that I satisfies all of the following properties:

1. $\Delta^I \subseteq N$;
2. $c^I = c$ for each individual $c \in N_{\mathcal{A}}$;
3. for each atomic role R, each pair in R^I is of the form $\langle s, s.i\rangle$, $\langle s.i, s\rangle$, or $\langle a, b\rangle$ for $s \in N$ and $a, b \in N_{\mathcal{A}}$;
4. for each pair $\langle c, d\rangle \in R^I$ such that $c, d \in N_{\mathcal{A}}$, we have $c = d$ or $R(c, d) \in \Xi_{\mathcal{T}}(\mathcal{A})$; and
5. for each atomic role R, each individual $c \in N_{\mathcal{A}}$ and each $c.i \in N$, if $\{\langle c, c.i\rangle, \langle c.i, c\rangle\} \subseteq R^I$, then there exist concepts A and B and a role S such that $A \sqsubseteq \exists S.B \in \mathcal{T}$, $S \sqsubseteq^*_{\mathcal{T}} R$, $S \sqsubseteq^*_{\mathcal{T}} R^-$, and $c \in A^I$.

A model I of $\Omega_{\mathcal{T}}$ satisfying properties (1)–(3) can be obtained, for example, using the hypertableau calculus by Motik *et al.* [2009]. Furthermore, if translated into first-order logic, all role atoms in the consequent of an axiom in $\Omega_{\mathcal{T}}$ are of the form $R(x, x)$, or they occur in formulae of the form $\exists y.R(x, y) \wedge \ldots$; thus, the hypertableau calculus cannot derive an atom of the form $R(a, b)$ with $a \neq b$, thus ensuring property (4). Finally, since $\Omega_{\mathcal{T}}$ is normalised, concepts of the form $\exists S.B$ occur in $\Omega_{\mathcal{T}}$ only in axioms of the form $A \sqsubseteq \exists S.B$; but then, the hypertableau calculus ensures that $\langle c, c.i\rangle \in S^I$ or $\langle c.i, c\rangle \in S^I$ only if $c \in A^I$; consequently, the only way for $\{\langle c, c.i\rangle, \langle c.i, c\rangle\} \subseteq R^I$ to hold is if property (5) holds.

To complete the proof, we next construct an interpretation J and show that $J \models \mathcal{T} \cup \mathcal{A}$ and $J \not\models \alpha$. In particular, let J be the following interpretation defined inductively on the quasi-ordering corresponding to relation $\sqsubseteq^*_{\mathcal{T}}$:

- $\Delta^J = \Delta^I$;
- $c^J = c^I = c$ for each individual $c \in N_{\mathcal{A}}$;
- $A^J = A^I$ for each atomic concept A;
- R^J is the transitive closure of R^I for each atomic role R that is transitive in $\mathcal{T}$; and
- $R^J = R^I \cup \bigcup_{S \sqsubseteq^*_{\mathcal{T}} R \text{ and } R \not\sqsubseteq^*_{\mathcal{T}} S} S^J$ for each atomic role R that is not transitive in $\mathcal{T}$.

If $\mathcal{T}$ does not contain concepts of the form $\exists R.\mathsf{self}$, then $J \models \mathcal{T}$ follows from the standard proofs of transitivity elimination in $\mathcal{SHI}$ [Simancik, 2012] and DL-Lite$^{\mathcal{H},+}_{bool}$ [Artale *et al.*, 2009]; furthermore, it is easy to see that the presence of atoms $\exists R.\mathsf{self}$ requires only minor changes to these proofs. Furthermore, since $\mathcal{A} \subseteq \Xi_{\mathcal{T}}(\mathcal{A})$, we clearly have $J \models \mathcal{A}$.

We are left to show that $J \not\models \alpha$. If α is of the form $A(c)$, the claim follows from the proofs by Simancik [2012] and Artale *et al.* [2009]. Hence, assume that α is of the form $\alpha = T(c, d)$, and assume for the sake of contradiction that $J \models T(c, d)$. Then, by the definition of J, there exist an atomic role R and $\{u_0, u_1, \ldots, u_n\} \subseteq \Delta^I$ such that R is transitive in $\mathcal{T}$, $R \sqsubseteq^*_{\mathcal{T}} T$, $c = u_0$, $d = u_n$, and $\langle u_{i-1}, u_i\rangle \in R^I$ for each $1 \leq i \leq n$. We consider the following two cases.

- Assume that, for each $0 \leq i \leq n$, if $u_i \in N_{\mathcal{A}}$, then $u_i = c$. Then, we clearly have $c = d$. Since I satisfies property (3), some $1 \leq i < n$ exists such that u_i is of the form $c.j$ for some j and $\{\langle c, c.j\rangle, \langle c.j, c\rangle\} \subseteq R^I$ holds. Furthermore, since I satisfies property (5), concepts A and B and a role S exist such that $A \sqsubseteq \exists S.B \in \mathcal{T}$, $S \sqsubseteq^*_{\mathcal{T}} R$, $S \sqsubseteq^*_{\mathcal{T}} R^-$, and $c \in A^I$. By Definition 5, then $A \sqsubseteq \exists R.\mathsf{self} \in \Omega_{\mathcal{T}}$, which implies $\langle c, c\rangle \in R^I$. Finally, $R \sqsubseteq^*_{\mathcal{T}} T$ implies $R \sqsubseteq^*_{\Omega_{\mathcal{T}}} T$; hence, we have $\langle c, c\rangle \in T^I$ as well, which contradicts our assumption that $I \not\models \alpha$.
- Assume that some $1 \leq i \leq n$ exists such that $u_i \in N_{\mathcal{A}}$ and $u_i \neq c$. We eliminate from the sequence $u_0, u_1, \ldots, u_n$ each subsequence $u_{i+1}, \ldots, u_j$ with $0 \leq i < j \leq n$ such that $u_i \in N_{\mathcal{A}}$, $u_j \in N_{\mathcal{A}}$, and $u_k \in N \setminus N_{\mathcal{A}}$ for each $i < k < j$; let $v_0, \ldots, v_\ell$ be the resulting sequence. Since I satisfies property (3), each eliminated subsequence satisfies $u_i = u_j$; hence, for each $1 \leq i \leq \ell$, we have $\langle v_{i-1}, v_i\rangle \in R^I$. Furthermore, since u_i exists such that $u_i \in N_{\mathcal{A}}$ and $u_i \neq c$, we have $\ell \geq 1$,

$v_0 = c$, and $v_\ell = d$. Finally, note that the above definition eliminates each subsequence u_i, u_{i+1} such that $u_i = u_{i+1}$ (condition $u_k \in N \setminus N_{\mathcal{A}}$ for each $i < k < j$ is then vacuously satisfied); therefore, sequence $v_0, \dots, v_\ell$ consists of distinct individuals in $N_{\mathcal{A}}$. But then, since I satisfies property (4), we have that $R(v_{i-1}, v_i) \in \Xi_{\mathcal{T}}(\mathcal{A})$ for each $1 \leq i \leq \ell$. Finally, by the definition of $\Xi_{\mathcal{T}}$, then $\Xi_{\mathcal{T}}(\mathcal{A})$ contains $R(v_0, v_\ell) = R(c, d)$, and consequently $T(c, d) \in \Xi_{\mathcal{T}}(\mathcal{A})$ as well. This, however, contradicts our assumption that $I \not\models \alpha$. □

C Proofs for Section 4.2

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.*

Sketch. The algorithm by Hustadt *et al.* [2007] first translates $\mathcal{T}$ into a set of skolemised clauses. An inspection of the algorithm reveals that, without concepts of the form $\exists R.\mathsf{self}$, each resulting clause is of one of the following forms, where R is an atomic role, f is a function symbol, and $A_{(i)}$, $B_{(i)}$, $C_{(i)}$, and $D_{(i)}$ are atomic concepts, $\top$, or $\bot$:

$$\neg A(x) \vee \underline{R(x, f(x))} \tag{20}$$
$$\neg A(x) \vee \underline{R(f(x), x)} \tag{21}$$
$$\underline{\neg R(x, y)} \vee S(x, y) \tag{22}$$
$$\underline{\neg R(x, y)} \vee S(y, x) \tag{23}$$
$$\neg A(x) \vee \underline{\neg R(x, y)} \vee \neg B(y) \vee C(x) \vee D(y) \tag{24}$$
$$\bigvee \neg A_i(x) \vee \bigvee \underline{\neg B_i(f(x))} \vee \bigvee C_i(x) \vee \bigvee \underline{D_i(f(x))} \tag{25}$$
$$\bigvee \underline{\neg A_i(x)} \vee \bigvee \underline{C_i(x)} \tag{26}$$

Furthermore, since $\mathcal{T}$ is normalised, axioms with concepts of the form $\exists R.\mathsf{self}$ are translated into clauses of the following form:

$$\neg \underline{R(x, x)} \vee A(x) \tag{27}$$
$$\neg A(x) \vee \underline{R(x, x)} \tag{28}$$

The algorithm next saturates the resulting set of clauses by *ordered* resolution, which is parameterised by a carefully constructed literal ordering and selection function; these parameters ensures that binary resolution and positive factoring are performed only with literals that are underlined in (20)–(28). The selection function can be extended to select atom $R(x, x)$ in each clause of type (27); furthermore, the ordering can be modified so that each atom $R(x, x)$ is larger than all atoms $A(x)$, thus ensuring that only atom $R(x, x)$ participates in inferences with clauses of type (28). Hustadt *et al.* [2007] show that each binary resolution or positive factoring inference, when applied to clauses of type (20)–(26), produces a clause of type (20)–(23) or (25)–(26). This is easily extended to clauses of type (27)–(28):

- a clause of type (27) cannot be resolved with any other clause;
- resolving a clause of type (28) with a clause of type (24) produces a clause of type (26); and
- resolving a clause of type (28) with a clause of type (22) or (23) produces a clause of type (28).

Hustadt *et al.* [2007] then show that the disjunctive program $\mathsf{DD}(\mathcal{T})$ can be obtained as the set of all clauses after saturation of type (22)–(24) and (26). For the case when $\mathcal{T}$ contains atoms of the form $\exists R.\mathsf{self}$, program $\mathsf{DD}(\mathcal{T})$ should also include clauses of type (27) and (28), and the proof by Hustadt *et al.* [2007] applies without any problems. Furthermore, it is straightforward to verify that $\mathsf{DD}(\mathcal{T})$ is a nearly-monadic program.

Finally, if $\mathcal{T}$ is a DL-Lite$^{\mathcal{H},+}_{bool}$-TBox, the only difference is that, in each clause of type (24), we have either $A = \top$ and $C = \bot$, or $B = \top$ and $D = \bot$. Since saturation does not introduce clauses of type (24), program $\mathsf{DD}(\mathcal{T})$ is clearly simple. □

D Proofs for Section 4.3

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 $\mathcal{N}_H \cup \mathcal{A} \models C$.*

Proof. To prove our claim, we assume that Procedure 1 is applied to $\mathcal{S} = \mathcal{N} \cup \mathcal{A}$. Towards this goal, we associate with each clause $C \in \mathcal{S}_H \cup \mathcal{S}_{\overline{H}}$ a set of facts F_C; for each such F_C, let $\neg F_C = \bigvee_{A \in F_C} \neg A$. We define F_C inductively on the applications of inference rules in Procedure 1; furthermore, we show in parallel that, at any point in time, for each clause $C \in \mathcal{S}_H \cup \mathcal{S}_{\overline{H}}$ and the corresponding set F_C, the following properties are satisfied:

(a) $\mathcal{N} \models \neg F_C \vee C$, and

(b) $F_C \subseteq \mathcal{A}$.

For the base case, consider an arbitrary clause $C \in \mathcal{S}$. If $C \in \mathcal{N}$, we define $F_C = \emptyset$; otherwise, we have $C \in \mathcal{A} \setminus \mathcal{N}$, so C is a fact, and we define $F_C = \{C\}$. Properties (a) and (b) are clearly satisfied.

For the induction step, assume that the two properties are satisfied for each clause $C \in \mathcal{S}_H \cup \mathcal{S}_{\overline{H}}$ at some point in time. We consider the following two ways in which Procedure 1 can extend $\mathcal{S}_H$ or $\mathcal{S}_{\overline{H}}$.

- Assume that resolution is applied to clauses $C_1 = D_1 \vee A_1$ and $C_2 = D_2 \vee \neg A_2$, deriving clause $C = D_1\sigma \vee D_2\sigma$. Let $F_C = F_{C_1} \cup F_{C_2}$, so property (b) is clearly satisfied. By induction assumption, we have $\mathcal{N} \models \neg F_{C_1} \vee D_1 \vee A_1$ and $\mathcal{N} \models \neg F_{C_2} \vee D_2 \vee \neg A_2$. By the soundness of binary resolution, we have $\{D_1 \vee A_1,\ \ D_2 \vee \neg A_2\} \models D_1\sigma \vee D_2\sigma$. But then, since $\neg F_{C_1}$ and $\neg F_{C_2}$ contain only constants, we have $\mathcal{N} \models \neg F_{C_1} \vee \neg F_{C_2} \vee D_1\sigma \vee D_2\sigma$, as required for (a).
- Assume that positive factoring is applied to a clause $C_1 = D_1 \vee A_1 \vee B_1$, deriving clause $C = D_1\sigma \vee A_1\sigma$. Let $F_C = F_{C_1}$, so property (b) is clearly satisfied. By induction assumption, we have $\mathcal{N} \models \neg F_{C_1} \vee D_1 \vee A_1 \vee B_1$. By the soundness of positive factoring, we have $\{D_1 \vee A_1 \vee B_1\} \models D_1\sigma \vee A_1\sigma$. But then, since $\neg F_{C_1}$ contains only constants, we have $\mathcal{N} \models \neg F_{C_1} \vee D_1\sigma \vee A_1\sigma$, as required for (a).

We now show the main claim of this theorem. To this end, consider an arbitrary Horn clause C such that $\mathcal{N} \cup \mathcal{A} \models C$. By Theorem 11, at some point in time during the application of Procedure 1 to $\mathcal{S}$, we have $\mathcal{S}_H \models C$. Note that $\mathcal{S}_H$ is a finite set.

Consider an arbitrary Horn clause $D \in \mathcal{S}_H$. By property (a), we have $\mathcal{N} \models \neg F_D \vee D$. Furthermore, $\neg F_D \vee D$ is a Horn clause, so by Theorem 11, at some point in time time during the application of Procedure 1 to $\mathcal{N}$, we have $\mathcal{N}_H^D \models \neg F_D \vee D$. Finally, by property (b), we have $F_D \subseteq \mathcal{A}$. These observations now imply that $\mathcal{N}_H^D \cup \mathcal{A} \models D$.

Now let $\mathcal{N}'_H = \bigcup_{D \in \mathcal{S}_H} \mathcal{N}_H^D$; clearly, $\mathcal{N}'_H \cup \mathcal{A} \models \mathcal{S}_H$. Note that Procedure 1 is monotonic in the sense that, if $\mathcal{N}_H \models E$ at some point in time for some clause E, then this also holds at all future points in time. Furthermore, $\mathcal{N}'_H$ is finite, so at some point in time during the application of Procedure 1 to $\mathcal{N}$, we have $\mathcal{N}_H \models \mathcal{N}'_H$. By the observations from the previous paragraph, we then have $\mathcal{N}_H \cup \mathcal{A} \models \mathcal{S}_H$ as well, which implies $\mathcal{N}_H \cup \mathcal{A} \models C$, as required. □

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.*

Proof. We prove by induction on the application of the inference rules in Procedure 1 that, at any point in time, $\mathcal{P}_H \cup \mathcal{P}_{\overline{H}}$ is a nearly-monadic program. The base case is clearly satisfied since $\mathcal{P}$ is nearly-monadic. For the induction base, we consider the possible inferences that can derive a clause in $\mathcal{P}_H \cup \mathcal{P}_{\overline{H}}$. First, note that positive factoring is never applicable to a clause of type 2 from Definition 7; furthermore, when applied to a clause of type 1, positive factoring always produces a clause of the same type. Second, since clauses of type 2 are Horn, binary resolution can be applied only if at least one clause is of type 1, and the resolvent is then clearly of type 1 as well. □

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 $\mathcal{T}$.*

Proof. Consider an arbitrary ABox $\mathcal{A}$ and an arbitrary fact α. By Theorem 6, we have that $\mathcal{T} \cup \mathcal{A} \models \alpha$ if and only if $\Omega_\mathcal{T} \cup \Xi_\mathcal{T}(\mathcal{A}) \models \alpha$. By Theorem 8, the latter holds if and only if $\mathsf{DD}(\Omega_\mathcal{T}) \cup \Xi_\mathcal{T}(\mathcal{A}) \models \alpha$. Moreover, since α is a Horn clause, by Theorem 12, the latter holds if and only if $\mathcal{P}_H \cup \Xi_\mathcal{T}(\mathcal{A}) \models \alpha$. We now show that the latter holds if and only if $\mathcal{P}_H \cup \Xi_\mathcal{T} \cup \mathcal{A} \models \alpha$. Clearly, $\mathcal{P}_H \cup \Xi_\mathcal{T}(\mathcal{A}) \models \alpha$ implies $\mathcal{P}_H \cup \Xi_\mathcal{T} \cup \mathcal{A} \models \alpha$ by monotonicity of first-order logic, so we next focus on showing that $\mathcal{P}_H \cup \Xi_\mathcal{T}(\mathcal{A}) \not\models \alpha$ implies $\mathcal{P}_H \cup \Xi_\mathcal{T} \cup \mathcal{A} \not\models \alpha$.

By Theorem 8, Lemma 13, and the fact that Procedure 1 is sound, program $\mathcal{P}_H$ is nearly-monadic and $\Omega_\mathcal{T} \models \mathcal{P}_H$. Now let $\mathcal{P}_H^m$ and $\mathcal{P}_H^r$ be the subsets of $\mathcal{P}_H$ of the rules of type 1 and 2, respectively. Since $\Omega_\mathcal{T} \models \mathcal{P}_H^r$, by the definition of $\Xi_\mathcal{T}$ we have $\Xi_\mathcal{T} \models \mathcal{P}_H^r$. Furthermore, if a role atom occurs in the head of a rule in $\mathcal{P}_H^m$, the atom is of the form $R(z, z)$; hence, each fact involving a role atom in $\mathcal{P}_H(\Xi_\mathcal{T}(\mathcal{A})) \setminus \Xi_\mathcal{T}(\mathcal{A})$ is necessarily of the form $R(c, c)$. But then, such facts clearly cannot trigger a transitivity rule in $\Xi_\mathcal{T}$ to derive a new fact; furthermore, for each rule $r \in \Xi_\mathcal{T}$ of the form $R(x, y) \to S(x, y)$ or $R(x, y) \to S(y, x)$, we have $\mathcal{P}_H^r \models r$; consequently, $\Xi_\mathcal{T}(\mathcal{P}_H(\Xi_\mathcal{T}(\mathcal{A}))) = \mathcal{P}_H(\Xi_\mathcal{T}(\mathcal{A}))$, and the property holds.

Thus, $\mathcal{T} \cup \mathcal{A} \models \alpha$ if and only if $\mathcal{P}_H \cup \Xi_\mathcal{T} \cup \mathcal{A} \models \alpha$ for arbitrary fact α; but then, for an arbitrary ground query Q, we also have $\mathcal{T} \cup \mathcal{A} \models Q$ if and only if $\mathcal{P}_H \cup \Xi_\mathcal{T} \cup \mathcal{A} \models Q$, as required. □

E Proofs for Section 4.4

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

Proof. Assume that Procedure 1 derives a clause C in line 5, and let D be the condensation of C. Since Procedure 1 is sound, we have $\mathcal{S}_H \cup \mathcal{S}_{\overline{H}} \models C$; furthermore, since C subsumes D, we have $\{C\} \models D$; but then, we have $\mathcal{S}_H \cup \mathcal{S}_{\overline{H}} \models D$ as well. It is therefore safe to add D to $\mathcal{S}_H$ or $\mathcal{S}_{\overline{H}}$, so let us assume that Procedure 1 does so; but then, this makes C redundant since D subsumes C by the definition of condensation. □

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

Proof. Let $\mathcal{P} = \mathcal{P}^m \cup \mathcal{P}^r$. Since $\mathcal{P}$ is simple, each rule in $\mathcal{P}^m$ is of the form (29) with each variable y_i occurring at most once in the rule, and each rule in $\mathcal{P}^r$ is of the form (30) or (31).

$$\bigwedge A_i(x) \wedge \bigwedge R_i(x,x) \wedge \bigwedge S_i(x,y_i) \wedge \bigwedge T_i(y_i,x) \rightarrow \bigvee U_i(x,x) \vee \bigvee B_i(x) \tag{29}$$

$$R(x,y) \rightarrow S(x,y) \tag{30}$$

$$R(x,y) \rightarrow S(y,x) \tag{31}$$

It is now straightforward to check that Procedure 1 derives only rules of such form: positive factoring is never applicable to a rule of the form (29)–(31), and binary resolution clearly derives only rules of these forms.

Now let C be an arbitrary rule derived in line 5 of Procedure 1, and let D be the condensation of C; furthermore, let n be the number of binary atoms occurring in $\mathcal{P}$. Since each variable in C occurs at most once in the rule, there can be at most $2n$ atoms of the form $R(x, y_i)$ or $R(y_i, x)$ different up to variable renaming; therefore, D contains at most $2n$ variables y_i. Since the number of predicates in D is linear in the size of $\mathcal{P}$, the size of each clause is linear in the size of $\mathcal{P}$ as well. But then, there can be at most exponentially many different clauses in $\mathcal{P}_H \cup \mathcal{P}_{\overline{H}}$, which implies termination of Procedure 1 using the standard argument [Hustadt *et al.*, 2007]. □

F Proofs for Section 5

We first present a well-known characterisation of the entailment of a datalog rule from a first-order theory. The proof of Proposition 23 is straightforward and can be found, for example, in the work by Cuenca Grau *et al.* [2012].

Proposition 23. *Let $\mathcal{F}$ be a set of first-order sentences, and let r be a datalog rule of the form $C_1 \wedge \ldots \wedge C_n \rightarrow H$. Then, for each substitution σ mapping each variable in r to a distinct individual not occurring in $\mathcal{F}$ or r, we have $\mathcal{F} \models r$ if and only if*

$$\mathcal{F} \cup \{\sigma(C_1), \ldots, \sigma(C_n)\} \models \sigma(H). \tag{32}$$

We are now ready to prove Theorem 22.

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.*

Proof. Let $Q = G(x_1)$ be a ground query, and let $\mathcal{T}$ be the $\mathcal{ELU}$-TBox corresponding to the program $\mathcal{P}$ from Example 16; thus, $\mathcal{T}$ consists of axioms (33)–(35), which are translated into disjunctive rules as shown below.

$$\top \sqsubseteq G \sqcup B \quad \rightsquigarrow \quad \top \rightarrow G(x) \vee B(x) \tag{33}$$

$$\exists E.G \sqsubseteq B \quad \rightsquigarrow \quad E(x_1,x_0) \wedge G(x_0) \rightarrow B(x_1) \tag{34}$$

$$\exists E.B \sqsubseteq G \quad \rightsquigarrow \quad E(x_1,x_0) \wedge B(x_0) \rightarrow G(x_1) \tag{35}$$

An individual v is *reachable* from an individual w by a path of length n in an ABox $\mathcal{A}$ if individuals $u_n, u_{n-1}, \ldots, u_0$ exist such that $E(u_i, u_{i-1}) \in \mathcal{A}$ for each $1 \leq i \leq n$, $u_n = v$, and $u_0 = w$. In this proof, we consider 0 to be an even number. We next prove the following property $(*)$, which characterises the answers to Q on $\mathcal{T} \cup \mathcal{A}$:

> For each ABox $\mathcal{A}$ containing only the E predicate and for each individual v, we have $v \in \mathsf{cert}(Q, \mathcal{T}, \mathcal{A})$ iff an individual w exists such that v is reachable from w by a path of positive even length and a path of positive odd length.

(Proof of $*$, direction $\Rightarrow$) Let v and w be arbitrary individuals such that v is reachable from w by a path of even length and a path of odd length; thus, $\mathcal{A}$ contains sets of assertions of the following form, where k is a positive even number, ℓ is a positive odd number, $u_k = u'_\ell = v$, and $u_0 = u'_0 = w$:

$$\{E(u_k, u_{k-1}), \ldots, E(u_1, u_0)\} \subseteq \mathcal{A} \tag{36}$$

$$\{E(u'_\ell, u'_{\ell-1}), \ldots, E(u'_1, u'_0)\} \subseteq \mathcal{A} \tag{37}$$

Let I be an arbitrary model of $\mathcal{T} \cup \mathcal{A}$. Due to axiom (33), we have the following two possibilities.

- Assume that $w \in G^I$. Then, axioms (34) and (35) and the assertions in (36) ensure that $u_j \in G^I$ for each even number $0 \leq j \leq k$ and $u_i \in B^I$ for each odd number $1 \leq i \leq k-1$; thus, we have $v \in G^I$. Furthermore, axioms (34) and (35) and the assertions in (37) ensure that $u'_i \in G^I$ for each even number $0 \leq i \leq \ell - 1$ and $u'_j \in B^I$ for each odd number $1 \leq j \leq \ell$; thus, we have $v \in B^I$. Consequently, we have $v \in B^I \cap G^I$.
- Assume that $w \in B^I$. By a symmetric argument we also conclude that $v \in B^I \cap G^I$.

Thus, we have $v \in B^I \cap G^I$ for an arbitrary model I of $\mathcal{T} \cup \mathcal{A}$, so $v \in \mathsf{cert}(Q, \mathcal{T}, \mathcal{A})$, as desired.

(Proof of $*$, direction $\Leftarrow$) Assume that $v \in \mathsf{cert}(Q, \mathcal{T}, \mathcal{A})$; furthermore, for the sake of contradiction assume that, for each individual w occurring in $\mathcal{A}$, each path from w to v in $\mathcal{A}$ is of odd length, or each path from w to v in $\mathcal{A}$ is of even length. Let I be the interpretation defined as follows:

- Δ^I contains all individuals in $\mathcal{A}$;
- $B^I = \{w \mid \text{each path from } w \text{ to } v \text{ in } \mathcal{A} \text{ is of even length}\} \cup \{v\}$;
- $G^I = \{w \mid \text{each path from } w \text{ to } v \text{ in } \mathcal{A} \text{ is of odd length}\}$; and
- $E^I = \{\langle c, d\rangle \mid E(c, d) \in \mathcal{A}\}$.

If there is no path from an individual w to individual v in $\mathcal{A}$, then each path from w to v in $\mathcal{A}$ is (vacuously) of both even and odd length, so $w \in B^I \cap G^I$; hence, axioms (33)–(35) are satisfied for such w. Furthermore, if w_1 is an individual such that each path from w_1 to v in $\mathcal{A}$ is of even length, and if w_2 satisfies the same property, then each path from w_1 to w_2 is also of even length; hence, axioms (33)–(35) are satisfied for such w_1 and w_2. Finally, if w_1 is an individual such that each path from w_1 to v in $\mathcal{A}$ is of odd length, and if w_2 satisfies the same property, then each path from w_1 to w_2 is also of even length; hence, axioms (33)–(35) are satisfied for such w_1 and w_2. Thus, have have $I \models \mathcal{T} \cup \mathcal{A}$; however, $v \notin G^I$, which is a contradiction.

This completes the proof of property $(*)$. Now let $\mathcal{P}$ be the following datalog program, where odd and $even$ are fresh binary predicates:

$$E(x_1, x_0) \rightarrow odd(x_1, x_0) \tag{38}$$
$$odd(x_2, x_1) \wedge E(x_1, x_0) \rightarrow even(x_2, x_0) \tag{39}$$
$$even(x_2, x_1) \wedge E(x_1, x_0) \rightarrow odd(x_2, x_0) \tag{40}$$
$$odd(x, y) \wedge even(x, y) \rightarrow G(x) \tag{41}$$
$$E(x_1, x_0) \wedge G(x_0) \rightarrow B(x_1) \tag{42}$$
$$E(x_1, x_0) \wedge B(x_0) \rightarrow G(x_1) \tag{43}$$

Furthermore, let $\mathcal{A}$ be an arbitrary ABox, and let $\mathcal{A}'$ be the subset of $\mathcal{A}$ containing precisely the assertions involving the E predicate. Due to rules (38)–(41), for each individual v we have $\mathcal{P} \cup \mathcal{A}' \models G(v)$ iff an individual w exists such that v is reachable from w in $\mathcal{A}'$ via an even and an odd path; by property $(*)$, the latter is the case iff $\mathcal{T} \cup \mathcal{A}' \models G(v)$. Rules (42) and (43) correspond to axioms (34) and (35), and they merely 'propagate' G and B from individuals explicitly labelled with G and B in $\mathcal{A}$; hence, it should be clear that $\mathcal{P}$ is a Q-rewriting of $\mathcal{T}$. Note, however, that $\mathcal{P}$ is not a strong Q-rewriting of $\mathcal{T}$: it contains fresh predicates odd and $even$, so $\mathcal{T} \not\models \mathcal{P}$.

To complete the proof, we next show that no strong Q-rewriting of $\mathcal{T}$ exists. To this end, let $\mathcal{R}$ be the infinite set containing rule (44) instantiated for each positive even number n.

$$E(x_n, x_0) \wedge E(x_n, x_{n-1}) \wedge \ldots \wedge E(x_1, x_0) \rightarrow G(x_n) \tag{44}$$

It is straightforward to see that $\mathcal{T} \models \mathcal{R}$: one can derive all such rules using resolution and factoring as shown in Example 16. We next prove that $\mathcal{R}$ satisfies the following two properties, which immediately imply the claim of this the theorem.

1. $\mathcal{P}' \models \mathcal{R}$ for each strong Q-rewriting $\mathcal{P}'$ of $\mathcal{T}$.
2. For each finite set of datalog rules $\mathcal{P}'$ such that $\mathcal{T} \models \mathcal{P}'$, we have $\mathcal{P}' \not\models \mathcal{R}$.

(Property 1) Assume by contradiction that a strong Q-rewriting $\mathcal{P}'$ of $\mathcal{T}$ exists such that $\mathcal{P}' \not\models \mathcal{R}$; then, there exist a rule $r \in \mathcal{R}$ such that $\mathcal{P}' \not\models r$. Let $C_1, \ldots, C_n$ be the body atoms of r, and note that the head atom of r is $Q = G(x_n)$. Since r is a datalog rule and $\mathcal{P}'$ is a set of first-order formulas, by Proposition 23, for each substitution σ mapping each variable in r to a distinct individual, we have $\mathcal{P}' \cup \{\sigma(C_1), \ldots \sigma(C_n)\} \not\models \sigma(Q)$. Now let σ be one such arbitrarily chosen substitution, and let $\mathcal{A} = \{\sigma(C_1), \ldots \sigma(C_n)\}$; clearly, we have $\mathcal{P}' \cup \mathcal{A} \not\models \sigma(Q)$. In contrast, $\mathcal{R} \cup \mathcal{A} \models \sigma(Q)$, and, due to $\mathcal{T} \models \mathcal{R}$, we have $\mathcal{T} \cup \mathcal{A} \models \sigma(Q)$. Thus, $\mathcal{P}'$ is not a strong Q-rewriting of $\mathcal{T}$, which contradicts our assumption.

(Property 2) Let $\mathcal{P}'$ be an arbitrary finite set of datalog rules such that $\mathcal{T} \models \mathcal{P}'$, let m be the maximal number of body atoms in a rule in $\mathcal{P}'$, let n be the smallest even number such that $n > m$, and let $\mathcal{A}$ be the following ABox where each v_i is distinct:

$$\mathcal{A} = \{E(v_n, v_0), E(v_n, v_{n-1}), E(v_{n-1}, v_{n-2}), \ldots, E(v_1, v_0)\} \tag{45}$$

We next show that, for each fact α, we have $\mathcal{P}' \cup \mathcal{A} \models \alpha$ iff $\alpha \in \mathcal{A}$; this clearly implies $\mathcal{P}' \cup \mathcal{A} \not\models G(v_n)$, which by Proposition 23 implies $\mathcal{P}' \not\models \mathcal{R}$, as required for Property 2. We proceed by contradiction, so assume that a fact α exists such that $\mathcal{P}' \cup \mathcal{A} \models \alpha$ and $\alpha \notin \mathcal{A}$. Then, a rule $r \in \mathcal{P}'$ of the form $r = C_1 \wedge \ldots \wedge C_k \rightarrow H$ and a substitution σ exist such that, for $\mathcal{A}' = \{\sigma(C_1), \ldots, \sigma(C_k)\}$, we have $\mathcal{A}' \subseteq \mathcal{A}$, $\alpha = \sigma(H)$, and $\alpha \notin \mathcal{A}$; note that $\mathcal{R} \cup \mathcal{A}' \models \alpha$. We now make the following observations.

- Since $\mathcal{T} \models \mathcal{P}'$, we have $\mathcal{T} \cup \mathcal{A}' \models \alpha$.
- Since $k \leq m < n$, we have $\mathcal{A}' \subsetneq \mathcal{A}$.
- Let $r' \in \mathcal{P}'$ be an arbitrary non-tautological rule of the form $r' = C'_1 \wedge \ldots \wedge C'_\ell \rightarrow H'$. Since $\mathcal{T} \models \mathcal{P}'$, we have $\mathcal{T} \models r'$. The latter, however, is possible only if H' is a unary atom involving the G or the B predicate, and each C'_i is an atom involving the G, B, or E predicate. Thus, either $\alpha = B(v_i)$ or $\alpha = G(v_i)$ for some integer i.

- By property $(*)$, we have $\mathcal{T} \cup \mathcal{A}' \not\models G(v_j)$ and $\mathcal{T} \cup \mathcal{A}' \not\models B(v_j)$ for each $n > j \geq 0$ since each such individual v_j is reachable from other individuals in $\mathcal{A}'$ by at most one path. Thus, we have $i = n$ in the previous item.
- Individual v_n is reachable from v_0 via two paths in $\mathcal{A}$; furthermore, due to $\mathcal{A}' \subsetneq \mathcal{A}$, individual v_n is reachable from v_0 in $\mathcal{A}'$ via at most one path. Therefore, by property $(*)$, we have $\mathcal{T} \cup \mathcal{A}' \not\models G(v_n)$ and $\mathcal{T} \cup \mathcal{A}' \not\models B(v_n)$.

The above four points are clearly in contradiction, which completes our proof. □