Polynomial Disjunctive Datalog Rewritings of Instance Queries in Expressive Description Logics

Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Šimkus

Institute of Information Systems, TU Vienna, Austria

Abstract. Rewriting ontology mediated queries (OMQs) into traditional query languages like FO-queries and Datalog is central for understanding their relative expressiveness, and plays a crucial role in the ongoing efforts to develop OMQ answering tools by reusing existing database technologies. However, the vast majority of works focus on Horn ontologies, and for OMQs where the ontologies are written in extensions of $\mathcal{ALC}$, only a couple of exponential rewritings into disjunctive Datalog are known. In this paper we propose a translation of instance queries mediated by ontologies in the expressive DL $\mathcal{ALCHI}$, into polynomial-sized disjunctive Datalog programs. The translation is based on a simple game-like algorithm, and can be extended to accommodate nominals. We can also rewrite OMQs with *closed-predicates* into Datalog programs with (limited) negation. Closed predicates are useful for combining complete and incomplete information, but make OMQs non-monotonic and thus not rewritable into positive disjunctive Datalog.

1 Introduction

In *ontology-mediated queries* (OMQs), a database query is enriched with an ontology, providing knowledge to obtain more complete answers from incomplete data. OMQs are receiving much attention in the database and knowledge representation research communities, particularly when the ontological knowledge is expressed in Description Logics (DLs) or in rule-based formalisms like *existential rules* and Datalog±, see e.g., [4, 3, 13] and their references. One of the most central questions in OMQ research is *rewritability*: given an OMQ Q, specified by a query and an ontology, obtain a query Q'—in some standard database language, like *first-order (FO) queries* or (fragments of) Datalog—such that, for any dataset $\mathcal{A}$, the certain answers to Q and Q' coincide. Such a Q' is called a *rewriting* of Q. Its existence and size are crucial for understanding the relative expressiveness of different families of OMQs in terms of more traditional query languages. Rewritings are also very relevant in practice, since they allow to reuse existing database technologies to support OMQ answering.

Research into OMQs that can be rewritten into *first-order (FO) queries* has produced the successful *DL-Lite* family [5] of DLs. The succinctness of rewritings for OMQs consisting of the so-called *(unions of) conjunctive queries (UCQs)* paired with ontologies in DL-Lite, or in related families of existential rules, has been extensively studied. It has been shown that polynomially sized non-recursive DATALOG queries can be constructed for a few well known ontology languages, formulated as DLs or existential rules [14]; but for many other cases, the lack of succinctness of such rewritings has also been established [11], and some authors have considered rewritings that, unlike the ones we consider here, are not *data-independent* [18, 12]. For DLs beyond *DL-Lite*, which are often not FO-rewritable, rewritings of UCQs into DATALOG queries have been proposed, and lie at the core of implemented systems, e.g., [24, 9, 25]. The pioneering work in [15] showed that instance queries in an expressive extension of $\mathcal{ALC}$ can be rewritten into a program in disjunctive DATALOG, using a constant number of variables per rule, but exponentially many rules. The first translation from CQs in expressive DLs ($\mathcal{SH}$, $\mathcal{SHQ}$) to disjunctive DATALOG programs was introduced in [8], but the program may contain double exponentially many predicates. For $\mathcal{ALC}$ and UCQs, the existence of exponential rewritings in disjunctive DATALOG was shown recently [4]. For restricted fragments of $\mathcal{SHI}$ and classes of CQs translations to DATALOG were investigated in [16, 17]. A polynomial time DATALOG translation of instance queries has been proposed in [23], but for a so-called *Horn-DL* that lacks disjunction. To our knowledge, this was until now the only polynomial rewriting for a DL that is not FO-rewritable.

The main contribution of this paper is a polynomial time translation of instance queries mediated by $\mathcal{ALCHI}$ TBoxes into *disjunctive* DATALOG. The translation is based on a simple game-theoretic characterization, and implements a so-called *type-elimination procedure* using a polynomially sized DATALOG program. To our knowledge, this is the first polynomial time translation of an expressive (non-Horn) DL into disjunctive DATALOG. The translation we present here can be extended to richer OMQs, and in particular, to *nominals* and *closed predicates*. In fact, we report here a simplified version of the translation in [1], which is a polynomial translation of instance queries mediated by $\mathcal{ALCHOI}$ TBoxes with closed predicates into disjunctive DATALOG queries with negation.

2 Preliminaries

We give some basic notions of DLs and DATALOG.

The DL $\mathcal{ALCHI}$ We assume countably infinite, mutually disjoint sets $\mathsf{N_R}$ of *role names*, $\mathsf{N_C}$ of *concept names*, and $\mathsf{N_I}$ of *individual names*. A *role* r is either a role name p, or an expression p^-, called the *inverse* of p. We let $r^- = p$ if $r = p^-$. *Concepts* are defined inductively: *(i)* $\top$, $\bot$ and all $A \in \mathsf{N_C}$ are concepts; *(ii)* if C_1, C_2 are concepts, then $C_1 \sqcap C_2$, $C_1 \sqcup C_2$ and $\neg C_1$ are concepts; *(iii)* if r is a role, and C is a concept, then $\exists r.C$, $\forall r.C$ are concepts. A *concept inclusion* is an expression of form $C_1 \sqsubseteq C_2$, where C_1, C_2 are concepts. A *role inclusion* is an expression of form $r_1 \sqsubseteq r_2$, where r_1, r_2 are roles. A *TBox* $\mathcal{T}$ is a finite set of (concept and role) inclusions. An *ABox* $\mathcal{A}$ is a finite set of *assertions* of the forms $A(a)$ and $p(a, b)$, where $\{a, b\} \subseteq \mathsf{N_I}$, $A \in \mathsf{N_C}$, and $p \in \mathsf{N_R}$. A *knowledge base (KB)* is a tuple $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox.

An *interpretation* is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a non-empty set (called the *domain*), $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for each $A \in \mathsf{N_C}$, $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for each $r \in \mathsf{N_R}$, and $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each $a \in \mathsf{N_I}$. The function $\cdot^{\mathcal{I}}$ is extended to the remaining concepts and roles in the standard way [2]. An interpretation $\mathcal{I}$ *satisfies* an inclusion $q_1 \sqsubseteq q_2$, if $q_1^{\mathcal{I}} \subseteq q_2^{\mathcal{I}}$, in symbols $\mathcal{I} \models q_1 \sqsubseteq q_2$; and it satisfies an assertion $q(\boldsymbol{a})$ if $(\boldsymbol{a})^{\mathcal{I}} \in q^{\mathcal{I}}$, in symbols, $\mathcal{I} \models q(\boldsymbol{a})$. For Γ a TBox or ABox, we write $\mathcal{I} \models \Gamma$ if $\mathcal{I} \models \alpha$ for all $\alpha \in \Gamma$. For a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we write $\mathcal{I} \models \mathcal{K}$ if the following hold: *(i)* $a \in \Delta^{\mathcal{I}}$ and $a^{\mathcal{I}} = a$ for each $a \in \mathsf{N_I}$ occurring in $\mathcal{K}$,[1] *(ii)* $\mathcal{I} \models \mathcal{T}$, and *(iii)* $\mathcal{I} \models \mathcal{A}$. For an assertion α, we write $\mathcal{K} \models \alpha$ if $\mathcal{I} \models \alpha$ for all $\mathcal{I}$ with $\mathcal{I} \models \mathcal{K}$.

Instance Queries An *(ontology mediated) instance query* is a tuple $Q = (\mathcal{T}, q)$, where $\mathcal{T}$ is a TBox, and $q \in \mathsf{N_C} \cup \mathsf{N_R}$. Let $\boldsymbol{a} \in \mathsf{N_I}$ in case $q \in \mathsf{N_C}$, and $\boldsymbol{a} \in \mathsf{N_I}^2$ otherwise. Then $\boldsymbol{a}$ is a *certain answer* to Q over an ABox $\mathcal{A}$ if $(\mathcal{T}, \mathcal{A}) \models q(\boldsymbol{a})$. To ease presentation, we assume that every concept and role of $\mathcal{A}$ also occurs in $\mathcal{T}$.

Normal Form Our results apply to arbitrary TBoxes, but to simplify presentation, we consider TBoxes in *normal form* where inclusions take one of the following forms:

$$\begin{array}{llll}(\mathbf{N1}) & A_1 \sqcap \dots \sqcap A_n \sqsubseteq A_{n+1} \sqcup \dots \sqcup A_k & (\mathbf{N3}) & A \sqsubseteq \forall r.A' \\ (\mathbf{N2}) & A \sqsubseteq \exists r.A' & (\mathbf{N4}) & r \sqsubseteq s\end{array}$$

with r, s roles, $\{A_1, \dots, A_k\} \subseteq \mathsf{N_C}$, and $\{A, A'\} \subseteq \mathsf{N_C}$. We also assume that $\mathcal{T}$ is closed under role inclusions as follows: (a) $p \sqsubseteq p \in \mathcal{T}$, for each $p \in \mathsf{N_R}$ occurring in $\mathcal{T}$, (b) if $r \sqsubseteq s \in \mathcal{T}$, then $r^- \sqsubseteq s^- \in \mathcal{T}$, and (c) if $r_1 \sqsubseteq r_2 \in \mathcal{T}$ and $r_2 \sqsubseteq r_3 \in \mathcal{T}$, then $r_1 \sqsubseteq r_3 \in \mathcal{T}$. For Γ a TBox, ABox, or KB, we denote by $\mathsf{N_I}(\Gamma)$, $\mathsf{N_R}(\Gamma)$, $\mathsf{N_C}(\Gamma)$ the set of individuals, role and concept names that occur in Γ, resp.

Disjunctive Datalog We assume countably infinite sets $\mathsf{N_P}$ and $\mathsf{N_V}$ of *predicate symbols* (each with an associated *arity*) and *variables*, respectively. We further assume that $\mathsf{N_C} \cup \mathsf{N_R} \subseteq \mathsf{N_P}$ with each $A \in \mathsf{N_C}$ being unary, and each $r \in \mathsf{N_R}$ being binary. An *atom* is an expression of the form $R(t_1, \dots, t_n)$, where $\{t_1, \dots, t_n\} \subseteq \mathsf{N_I} \cup \mathsf{N_V}$, and R is an n-ary relation symbol. A *rule* ρ is an expression of the form $h_1 \vee \ldots \vee h_n \leftarrow b_1, \dots, b_k$, where $H = \{h_1, \dots, h_n\}$ and $B = \{b_1, \dots, b_k\}$ are sets of atoms, called the *head* and the *body* of ρ, respectively. Each variable that occurs in H must also occur in B. Rules of the form $h \leftarrow$ (known as *facts*) are simply identified with the atom h, thus ABox assertions are valid facts in our syntax. For a role name p, we may use $p^-(t_1, t_2)$ to denote the atom $p(t_2, t_1)$. A *program* is any finite set P of rules. We use $\mathsf{ground}(P)$ to denote the *grounding* of P, i.e. the variable-free program that is obtained from P by applying on its rules all the possible substitutions of variables by individuals of P. An *(Herbrand) interpretation* (or, *database*) I is any finite set of variable-free (or, *ground*) atoms. An interpretation I is a *model* of a program P if $\{b_1, \dots, b_k\} \subseteq I$ implies $I \cap \{h_1, \dots, h_n\} \neq \emptyset$ for all rules $h_1 \vee \ldots \vee h_n \leftarrow b_1, \dots, b_k$ in $\mathsf{ground}(P)$. A *(*Datalog*) query* is a pair (P, q), where P is a program, and q is a predicate

[1] This is called the *standard name assumption (SNA)*, which we apply to the individuals occurring in $\mathcal{K}$.

symbol from P. A tuple $\boldsymbol{a}$ of constants is a *certain answer to* (P, q) *over a database* I if $q(\boldsymbol{a}) \in J$ for all models J of $P \cup I$.

3 Characterization of Counter Models

Assume a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and an assertion β. Towards deciding $\mathcal{K} \not\models \beta$ using a polynomially sized program, we decompose the problem into two steps:

(1) Guess a *core interpretation* $\mathcal{I}_c$ for $\mathcal{K}$, whose domain is $\mathsf{N_I}(\mathcal{A})$. Core interpretations fix how the individuals of $\mathcal{K}$ participate in concepts and roles, ensuring the satisfaction of $\mathcal{A}$, and the non-entailment of β.
(2) Check that $\mathcal{I}_c$ can be extended to satisfy all axioms in $\mathcal{T}$.

Defining DATALOG rules that do (1) is not hard, but (2) is more challenging, and will rely on a game-theoretic characterization we describe below. But first we need to define core interpretations.

Definition 1. *A* core interpretation *for a KB* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ *is any interpretation* $\mathcal{I}_c$ *such that*

(c1) $\Delta^{\mathcal{I}_c} = \mathsf{N_I}(\mathcal{A})$ *and* $a^{\mathcal{I}_c} = a$ *for all* $a \in \mathsf{N_I}(\mathcal{A})$*,*
(c2) $\mathcal{I}_c \models \mathcal{A}$*,*
(c3) $\mathcal{I}_c \models A \sqsubseteq \forall r.A'$ *for all* $A \sqsubseteq \forall r.A' \in \mathcal{T}$*,*
(c4) $\mathcal{I}_c \models r \sqsubseteq s$ *for all* $r \sqsubseteq s \in \mathcal{T}$*,*
(c5) $q^{\mathcal{I}_c} = \emptyset$ *for each* $q \notin \mathsf{N_C}(\mathcal{T}) \cup \mathsf{N_R}(\mathcal{T})$*.*

An interpretation $\mathcal{J}$ *is called an* extension *of* $\mathcal{I}_c$*, if* $\mathcal{I}_c$ *is the result of restricting* $\mathcal{J}$ *to* $\mathsf{N_I}(\mathcal{A})$*.*

A core and its extensions coincide on the assertions they entail, and deciding non-entailment of an instance query q amounts to deciding whether there is a core that does not entail q, and that can be extended into a model. But verifying whether a core can be extended into a full model is hard: it corresponds to testing consistency (of $\mathcal{I}_c$ viewed as an ABox) with respect to $\mathcal{T}$, an EXPTIME-hard problem already for fragments of $\mathcal{ALCHI}$. In order to obtain a polynomial set of rules that solves this EXPTIME-hard problem, we characterize it as a game, revealing a simple algorithm that admits an elegant implementation in DATALOG.

Definition 2. *A* type (over a TBox $\mathcal{T}$) τ *is a subset of* $\mathsf{N_C}(\mathcal{T}) \cup \{\top\}$ *such that* $\bot \notin \tau$ *and* $\top \in \tau$*. We say that* τ satisfies *an inclusion* $\alpha = A_1 \sqcap \cdots \sqcap A_n \sqsubseteq A_{n+1} \sqcup \cdots \sqcup A_k$ *of type* (**N1**)*, if* $\{A_1, \ldots, A_n\} \subseteq \tau$ *implies* $\{A_{n+1}, \ldots, A_k\} \cap \tau \neq \emptyset$*; otherwise* τ *violates* α*. For an element* $e \in \Delta^{\mathcal{I}}$ *in an interpretation* $\mathcal{I}$*, we let* $type(e, \mathcal{I}) = \{B \in \mathsf{N_C} \mid e \in B^{\mathcal{I}}\}$*. A type* τ *is* realized *in* $\mathcal{I}$ *if there is some* $e \in \Delta^{\mathcal{I}}$ *s.t.* $type(e, \mathcal{I}) = \tau$*.*

We now describe a game to decide whether a given core $\mathcal{I}_c$ can be extended into a model of a KB $\mathcal{K}$. The game is played by Bob (the builder), who wants to extend $\mathcal{I}_c$ into a model, and Sam (the spoiler), who wants to spoil all Bob's attempts. Sam starts by picking an individual a, and they look at its type $\mathit{type}(a,\mathcal{I}_c)$. If it doesn't satisfy the inclusions of type (**N1**) Sam wins. Otherwise, in each turn Sam chooses an inclusion of the form $A \sqsubseteq \exists r A'$ which would need to be satisfied by (an element with) the current type, forcing Bob to pick a type for the corresponding r-successor that satisfies A'. The game continues for as long as Bob can respond to the challenges of Sam.

The game on $\mathcal{I}$ starts by Sam choosing an individual $a \in \Delta^{\mathcal{I}}$, and $\tau = \mathit{type}(a,\mathcal{I})$ is set to be the *current type*. Then:

($\blacklozenge$) If τ does not satisfy all inclusions of type (**N1**) in $\mathcal{T}$, then Sam is declared the winner. Otherwise, Sam chooses an inclusion $A \sqsubseteq \exists r.A' \in \mathcal{T}$ with $A \in \tau$; if there is no such inclusion, Bob is declared the winner. Otherwise, Bob chooses a new type τ' such that:

(C1) $A' \in \tau'$, and

(C2) for all inclusions $A_1 \sqsubseteq \forall s.A_2 \in \mathcal{T}$:

- if $r \sqsubseteq s \in \mathcal{T}$ and $A_1 \in \tau$ then $A_2 \in \tau'$,
- if $r^- \sqsubseteq s \in \mathcal{T}$ and $A_1 \in \tau'$ then $A_2 \in \tau$.

τ' is set to be the current type, and the game continues with a new round, i.e. we go back to $\blacklozenge$.

A *run* of the game on $\mathcal{I}$ is a (possibly infinite) sequence

$$a\alpha_1\tau_1\alpha_2\tau_2\ldots$$

where a is the individual picked initially by Sam, and each α_i and τ_i are the inclusion picked by Sam and the type picked by Bob in round i, respectively. A *strategy for Bob* is a partial function str that maps pairs of a type τ and an inclusion $A \sqsubseteq \exists r.A'$ with $A \in \tau$ to a type τ' that satisfies (C1) and (C2); intuitively, it gives a move for Bob in response to moves of Sam. A run $a\alpha_1\tau_1\alpha_2\tau_2\ldots$ with $\mathit{type}(a,\mathcal{I}) = \tau_0$ *follows a strategy* str if $\tau_i = \mathit{str}(\tau_{i-1},\alpha_i)$ for every $i \geq 1$.

For a finite run w, we let $\mathit{tail}(w) = \mathit{type}(a,\mathcal{I})$ if $w = a$, and $\mathit{tail}(w) = \tau_\ell$ if $w = a\ldots\alpha_\ell\tau_\ell$ with $\ell \geq 1$. The strategy str is called *non-losing* on $\mathcal{I}$ if for every finite run w that follows str, $\mathit{tail}(w)$ satisfies all inclusions of type (**N1**) in $\mathcal{T}$, and $\mathit{str}(\mathit{tail}(w), A \sqsubseteq \exists r.A')$ is defined for every $A \sqsubseteq \exists r.A' \in \mathcal{T}$ with $A \in \mathit{tail}(w)$.

Theorem 1. *Assume a KB $\mathcal{K}$ and an assertion β. Then $\mathcal{K} \not\models \beta$ iff there is a core interpretation $\mathcal{I}_c$ for $\mathcal{K}$ such that:*

(1) $\mathcal{I}_c \not\models \beta$, and
(2) there is a non-losing strategy for Bob on $\mathcal{I}_c$.

Proof. (Sketch.) We focus on showing that there is a non-losing strategy str for Bob on $\mathcal{I}_c$ iff there exists an extension $\mathcal{J}$ of $\mathcal{I}_c$ s.t. $\mathcal{J} \models \mathcal{K}$. The claim follows from this, and the easy claim that extensions preserve non-entailment of β.

For the "$\Rightarrow$" direction, from an arbitrary non-losing str for $\mathcal{I}_c$, we build $\mathcal{J}$ as follows. We denote by $\mathit{frn}(\mathcal{I}_c,\mathit{str})$ the set of all finite runs $a\alpha_1\tau_1\ldots\alpha_i\tau_i$, $i \geq 0$ that follow str. The domain of $\mathcal{J}$ is:

$$\Delta^{\mathcal{J}} = \mathit{frn}(\mathcal{I}_c,\mathit{str})$$

and for each $a \in \mathsf{N_I}$, each $A \in \mathsf{N_C}$, and each $p \in \mathsf{N_R}$ we let:

$$\begin{aligned} a^{\mathcal{J}} &= a^{\mathcal{I}_c} & A^{\mathcal{J}} &= \{w \mid A \in \mathit{tail}(w)\} \\ p^{\mathcal{J}} &= p^{\mathcal{I}_c} \cup \{(w, w\alpha_i\tau_i) \mid r_i \sqsubseteq p \in \mathcal{T}\} \cup \\ &\qquad \{(w\alpha_i\tau_i, w) \mid r_i^- \sqsubseteq p \in \mathcal{T}\} \end{aligned}$$

where $\alpha_i = A \sqsubseteq \exists r_i.A' \in \mathcal{T}$ and $\{w, w\alpha_i\tau_i\} \subseteq \Delta^{\mathcal{J}}$.

It is not hard to see that $\mathcal{J}$ is an extension of $\mathcal{I}_c$. We show that $\mathcal{J}$ is a model of $\mathcal{K}$. First, $\mathcal{J} \models \mathcal{A}$ by definition of $\mathcal{I}_c$ and the fact that $\mathcal{J}$ is an extension of $\mathcal{I}_c$. It is left to prove that $\mathcal{J}$ satisfies all inclusions in $\mathcal{T}$. Note that for each $w \in \Delta^{\mathcal{J}}$, $\mathit{type}(w, \mathcal{J}) = \mathit{tail}(w)$. That $\mathcal{J}$ satisfies all inclusions of type (**N1**) holds by definition of non-losing *str* and the fact that $\mathit{tail}(w)$ satisfies all such inclusions. For the inclusions $\alpha = A \sqsubseteq \exists r.A'$ of type (**N2**), we see that for each $w \in \Delta^{\mathcal{J}}$, if $w \in A^{\mathcal{J}}$, then $A \in \mathit{tail}(w)$ and w is a run that follows *str*. Hence, since *str* is a non-losing strategy $\mathit{str}(\mathit{tail}(w), \alpha)$ is defined, that is there exists a τ' over $\mathcal{T}$ such that $\mathit{str}(\mathit{tail}(w), \alpha) = \tau'$ and $A' \in \tau'$. Thus $w\alpha\tau'$ is a run that follows *str*, i.e. $w\alpha\tau' \in \mathit{frn}(\mathcal{I}_c, \mathit{str})$, so $w\alpha\tau'$ is in $A'^{\mathcal{J}}$ and $(w, w\alpha\tau') \in r^{\mathcal{J}}$ if r is a role name, and $(w\alpha\tau', w) \in r^{\mathcal{J}}$ otherwise. For the inclusions $A_1 \sqsubseteq \forall r.A_2$ of type (**N3**), we distinguish the case of $(a, a') \in r^{\mathcal{I}_c}$ for a pair a, a' of individuals in $\Delta^{\mathcal{I}_c}$, for which $a \in A_1^{\mathcal{J}}$ implies $a' \in A_2^{\mathcal{J}}$ holds by definition of $\mathcal{I}_c$, and the case of an arbitrary pair of objects $(w, w') \in r^{\mathcal{I}_c}$ where w' is a child of w (that is, w' is of the form $w\alpha_i\tau_i$), or vice-versa, for which $w' \in A_2^{\mathcal{J}}$ whenever $w \in A_1^{\mathcal{J}}$ is ensured by the condition (C2) in the definition of the game, the fact that *str* is a non-losing strategy, and the construction of $\mathcal{J}$. Finally, the inclusions of type (**N4**) are also guaranteed by the definition of the core $\mathcal{I}_c$ for pairs of individual names, and by the condition (C2) in the definition of the game for arbitrary pairs of an object and its child.

For "$\Leftarrow$", assume an arbitrary model $\mathcal{J}$ of $\mathcal{K}$ that is an extension of $\mathcal{I}_c$. We can extract from it a non-losing strategy for Bob as follows. First, let T be a the set of all the types realized in $\mathcal{J}$, and observe that each type in T satisfies all inclusions of type (**N1**) and that for all $a \in \mathsf{N_I}(\mathcal{A})$, $\mathit{type}(a, \mathcal{J}) \in T$. Then it suffices to set, for each $\tau \in T$ and each $A \sqsubseteq \exists r.A' \in \mathcal{T}$ with $A \in \tau$, $\mathit{str}(\tau, A \sqsubseteq \exists r.A') = \tau'$ for an arbitrarily chosen $\tau' \in T$ that satisfies (C1) and (C2), which exists because $\mathcal{J}$ is a model, hence it satisfies all existential and universal inclusions in $\mathcal{T}$, and all types realized in $\mathcal{J}$ are contained in T. This *str* is a non-losing strategy for Bob. $\square$

To decide whether Bob has a non-losing strategy on a given core we use the type elimination procedure **Mark** in Algorithm 1, which *marks* (or, *eliminates*) all types from which Sam has a winning strategy. It takes as input the TBox $\mathcal{T}$, and an interpretation $\mathcal{I}$ which intuitively is the core being checked. The algorithm starts by building the set N of all possible types over $\mathcal{T}$, and then it eliminates bad choices (from which Bob would loose) by marking them. In step ($\mathrm{M_{N1}}$), the algorithm marks in N all types that violate some inclusion of type (**N1**) in $\mathcal{T}$; Sam wins already in the first round on these types. Then, in the loop, ($\mathrm{M}_\exists$) exhaustively marks types τ that allow Sam to pick an inclusion $A \sqsubseteq \exists r.A'$ for

Algorithm 1: Mark

```
input   : TBox $\mathcal{T}$, interpretation $\mathcal{I}$
output  : Set of (possibly marked) types
$N \leftarrow \{\tau \mid \tau$ is a type over $\mathcal{T}\}$
($\mathrm{M_{N1}}$) Mark each $\tau \in N$ that violates some inclusion of the form (N1) in $\mathcal{T}$
repeat
    ($\mathrm{M}_\exists$) Mark each $\tau \in N$ such that $A \sqsubseteq \exists r.A' \in \mathcal{T}$, $A \in \tau$, and for each $\tau' \in N$,
    at least one the following holds:
      (C0)   $\tau'$ is marked,
      (C1')  $A' \notin \tau'$, or
      (C2')  there exists $A_1 \sqsubseteq \forall s.A_2 \in \mathcal{T}$ with
        – $r \sqsubseteq s \in \mathcal{T}$ and $A_1 \in \tau$ and $A_2 \notin \tau'$, or
        – $r^- \sqsubseteq s \in \mathcal{T}$ and $A_1 \in \tau'$ and $A_2 \notin \tau$
until no new type is marked
return $N$
```

which Bob cannot reply with any τ'. At the end the marked types, are those from which Bob can lose, so we get:

Theorem 2. *Let $\mathcal{I}_c$ be a core interpretation. Then Bob has a non-losing strategy on $\mathcal{I}_c$ iff none of the types realized in $\mathcal{I}_c$ is marked by* **Mark**$(\mathcal{T}, \mathcal{I}_c)$.

Proof. (Sketch.) For the "$\Rightarrow$" direction, we can show (by induction in the number of iterations of **Mark**$(\mathcal{T}, \mathcal{I}_c)$) that if a type is marked, then it cannot occur in a non-losing *str* for Bob. For the "$\Leftarrow$" direction, a non-losing *str* for $\mathcal{I}_c$ is obtained by taking all unmarked $\tau \in N$, and for each of them, and each $A \sqsubseteq \exists r.A' \in \mathcal{T}$ with $A \in \tau$, setting $str(\tau, A \sqsubseteq \exists r.A') = \tau'$ for an arbitrary unmarked τ' that satisfies (C1) and (C2). □

4 Translation into Datalog

Assume an instance query $(\mathcal{T}, q)$. We build next a polynomially sized program P such that the queries $(\mathcal{T}, q)$ and (P, q) have the same certain answers for all ABoxes over the signature of $\mathcal{T}$. Roughly, the program P has 3 major components: (a) rules to non-deterministically generate a core interpretation $\mathcal{I}_c$ for the KB $(\mathcal{T}, \mathcal{A})$, where $\mathcal{A}$ is an input ABox; (b) rules that implement the type elimination algorithm presented in the previous section; (c) rules that glue (a) and (b) together, ensuring that all types that occur in $\mathcal{I}_c$ are not marked by the marking procedure. We remark that the construction of P is independent from any particular ABox.

(I) Collecting the individuals We first add rules to collect in the unary predicate ind all the individuals that occur in the input ABox. For each $A \in \mathsf{N_C}(\mathcal{T})$, $r \in \mathsf{N_R}(\mathcal{T})$, we have:

$$\mathsf{ind}(x) \leftarrow A(x) \qquad \mathsf{ind}(x) \leftarrow r(x, y) \qquad \mathsf{ind}(y) \leftarrow r(x, y)$$

(II) Generating core interpretations For each $A \in \mathsf{N_C}(\mathcal{T})$ (resp., $r \in \mathsf{N_R}(\mathcal{T})$) we will use a fresh concept name $\overline{\mathsf{A}}$ (resp., role name $\overline{\mathsf{r}}$). We add the following rules to P:

$$\begin{array}{rcll} A(x) \vee \overline{\mathsf{A}}(x) & \leftarrow & \mathsf{ind}(x) & A \in \mathsf{N_C}(\mathcal{T}) \\ & \leftarrow & A(x), \overline{\mathsf{A}}(x) & A \in \mathsf{N_C}(\mathcal{T}) \\ r(x,y) \vee \overline{\mathsf{r}}(x,y) & \leftarrow & \mathsf{ind}(x), \mathsf{ind}(y) & r \in \mathsf{N_R}(\mathcal{T}) \\ & \leftarrow & r(x,y), \overline{\mathsf{r}}(x,y) & r \in \mathsf{N_R}(\mathcal{T}) \end{array}$$

To ensure *(c3)* in Definition 1, for each $A \sqsubseteq \forall r.A' \in \mathcal{T}$ we add the rule $A'(y) \leftarrow A(x), r(x,y)$. Then, to ensure *(c4)*, for each role inclusion $r \sqsubseteq s \in \mathcal{T}$ we add the rule $s(x,y) \leftarrow r(x,y)$.

Intuitively, the stable models of the above rules generate the different core interpretations $\mathcal{I}_c$ of the KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ for any given $\mathcal{A}$.

We next implement the algorithm **Mark** from Section 3. To obtain a polynomially sized program, we need to use non-ground rules whose number of variables depends on the number of different concept names in $\mathcal{T}$. Assume an arbitrary enumeration $A_1, \ldots, A_k$ of $\mathsf{N_C}(\mathcal{T})$. Assume also a pair $0, 1$ of special individuals. Intuitively, we will use a k-ary relation $\mathsf{Type} = \{0,1\}^k$ to store the set of all types over $\mathcal{T}$. Naturally, a k-tuple $(a_1, \ldots, a_k) \in \mathsf{Type}$ encodes the type $\tau = \{A_i \mid a_i = 1, 1 \leq i \leq k\} \cup \{\top\}$. We are most interested in computing a k-ary relation $\mathsf{Marked} \subseteq \{0,1\}^k$ that contains precisely the types marked by the **Mark** algorithm. We next define the rules to compute Type and Marked, and other relevant relations.

(III) A linear order over types The first ingredient is a linear order over all possible types. To this end, for every $1 \leq i \leq k$, we inductively define i-ary relations first^i and last^i, and a $2i$-ary relation next^i, which will provide the first, the last and the successor elements from a linear order on $\{0,1\}^i$. In particular, given $\boldsymbol{u}, \boldsymbol{v} \in \{0,1\}^i$, the fact $\mathsf{next}^i(\boldsymbol{u}, \boldsymbol{v})$ will be true if $\boldsymbol{v}$ follows $\boldsymbol{u}$ in the ordering of $\{0,1\}^i$. The rules to populate next^i are quite standard (see, e.g., Theorem 4.5 in [6]). For the case $i = 1$, we simply add the following facts:

$$\mathsf{first}^1(0) \leftarrow \qquad \mathsf{last}^1(1) \leftarrow \qquad \mathsf{next}^1(0,1) \leftarrow$$

Then, for all $1 \leq i \leq k-1$ we add the following rules:

$$\begin{array}{rcl} \mathsf{next}^{i+1}(0, \boldsymbol{x}, 0, \boldsymbol{y}) & \leftarrow & \mathsf{next}^i(\boldsymbol{x}, \boldsymbol{y}) \\ \mathsf{next}^{i+1}(1, \boldsymbol{x}, 1, \boldsymbol{y}) & \leftarrow & \mathsf{next}^i(\boldsymbol{x}, \boldsymbol{y}) \\ \mathsf{next}^{i+1}(0, \boldsymbol{x}, 1, \boldsymbol{y}) & \leftarrow & \mathsf{last}^i(\boldsymbol{x}), \mathsf{first}^i(\boldsymbol{y}) \\ \mathsf{first}^{i+1}(0, \boldsymbol{x}) & \leftarrow & \mathsf{first}^i(\boldsymbol{x}) \\ \mathsf{last}^{i+1}(1, \boldsymbol{x}) & \leftarrow & \mathsf{last}^i(\boldsymbol{x}) \end{array}$$

We can now collect in the k-ary relation Type all types over $\mathcal{T}$ (thus computing the set N of the **Mark** algorithm):

$$\mathsf{Type}(\boldsymbol{x}) \leftarrow \mathsf{first}^k(\boldsymbol{x}) \qquad \mathsf{Type}(\boldsymbol{y}) \leftarrow \mathsf{next}^k(\boldsymbol{x}, \boldsymbol{y})$$

(IV) Implementing Step ($\mathbf{M}_{\mathrm{N1}}$) First, we add the auxiliary facts $\mathsf{F}(0) \leftarrow$ and $\mathsf{T}(1) \leftarrow$ to P. For a k-tuple of variables $\boldsymbol{x}$, we let $B \in \boldsymbol{x}$ denote the atom $T(x_j)$, where j is the index of B in the enumeration of $\mathsf{N_C}(\mathcal{T})$. Similarly, we let $B \notin \boldsymbol{x}$ denote the atom $F(x_j)$, where j is the index of B in the enumeration. Then the step (M_{N1}), which marks types violating inclusions of type (**N1**), is implemented using the following rule for every inclusion $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq A'_1 \sqcup \cdots \sqcup A'_m$ of $\mathcal{T}$:

$$\mathsf{Marked}(\boldsymbol{x}) \leftarrow \mathsf{Type}(\boldsymbol{x}), A_1 \in \boldsymbol{x}, \ldots, A_n \in \boldsymbol{x}, A'_1 \notin \boldsymbol{x}, \ldots, A'_m \notin \boldsymbol{x}$$

(V) Implementing Step ($\mathbf{M}_{\exists}$) The following rules are added for each inclusion $\alpha = A \sqsubseteq \exists r.A' \in \mathcal{T}$. Recall that we need to mark a type τ if $A \in \tau$, and *for each* type τ' at least one of (C0), (C1′) or (C2′) holds. We first use an auxiliary $(2k+1)$ relation $\mathsf{MarkedOne}$ to collect all such types τ'.

Assume a fresh individual a_α. For collecting each τ' that satisfies (C0), we add:

$$\mathsf{MarkedOne}(\boldsymbol{x}, a_\alpha, \boldsymbol{y}) \leftarrow \mathsf{Type}(\boldsymbol{x}), \mathsf{Marked}(\boldsymbol{y})$$

For the condition (C1′), we add the rule:

$$\mathsf{MarkedOne}(\boldsymbol{x}, a_\alpha, \boldsymbol{y}) \leftarrow \mathsf{Type}(\boldsymbol{x}), \mathsf{Type}(\boldsymbol{y}), A' \notin \boldsymbol{y}$$

The rules for (C2′) are as follows. For all inclusions $A_1 \sqsubseteq \forall r.A_2 \in \mathcal{T}$ with $r \sqsubseteq s \in \mathcal{T}$, we add the rule

$$\mathsf{MarkedOne}(\boldsymbol{x}, a_\alpha, \boldsymbol{y}) \leftarrow \mathsf{Type}(\boldsymbol{x}), \mathsf{Type}(\boldsymbol{y}), A_1 \in \boldsymbol{x}, A_2 \notin \boldsymbol{y}$$

For all $A_1 \sqsubseteq \forall r.A_2 \in \mathcal{T}$ with $r^- \sqsubseteq s \in \mathcal{T}$, we also add

$$\mathsf{MarkedOne}(\boldsymbol{x}, a_\alpha, \boldsymbol{y}) \leftarrow \mathsf{Type}(\boldsymbol{x}), \mathsf{Type}(\boldsymbol{y}), A_1 \in \boldsymbol{y}, A_2 \notin \boldsymbol{x}$$

We infer $\mathsf{Marked}(\boldsymbol{t})$ in case $\mathsf{MarkedOne}(\boldsymbol{t}, a_\alpha, \boldsymbol{v})$ is true for all types (bit vectors) $\boldsymbol{v}$. To this end, we need another $(2k+1)$-ary relation $\mathsf{MarkedUntil}$. We add:

$$\begin{aligned}
\mathsf{MarkedUntil}(\boldsymbol{x}, a_\alpha, \boldsymbol{z}) \leftarrow& \mathsf{MarkedOne}(\boldsymbol{x}, a_\alpha, \boldsymbol{z}), \mathsf{first}^k(\boldsymbol{z}) \\
\mathsf{MarkedUntil}(\boldsymbol{x}, a_\alpha, \boldsymbol{u}) \leftarrow& \mathsf{MarkedUntil}(\boldsymbol{x}, a_\alpha, \boldsymbol{z}), \mathsf{next}^k(\boldsymbol{z}, \boldsymbol{u}), \\
& \mathsf{MarkedOne}(\boldsymbol{x}, a_\alpha, \boldsymbol{u})
\end{aligned}$$

Intuitively, with the above rules we traverse all types checking the conditions (C0), (C1′), (C2′) described in ($\mathrm{M}_\exists$). If we manage to reach the last type, we know the condition is satisfied. We add the following rule:

$$\mathsf{Marked}(\boldsymbol{x}) \leftarrow \mathsf{MarkedUntil}(\boldsymbol{x}, a_\alpha, \boldsymbol{z}), A \in \boldsymbol{x}, \mathsf{last}^k(\boldsymbol{z})$$

(VI) Forbidding marked types in the core We need to forbid each individual in the generated core interpretation from having a type from Marked. For all $0 \leq i \leq k$, we take a fresh $(i+1)$-ary relation symbol Proj^i. We first add:

$$\mathsf{Proj}^k(x, \boldsymbol{y}) \leftarrow \mathsf{ind}(x), \mathsf{Marked}(\boldsymbol{y})$$

We will now project away bits from the Proj^i relations by looking at the actual types of individuals. For all $1 \leq i \leq k$ we have the following rules:

$$\begin{aligned}\mathsf{Proj}^{i-1}(x, \boldsymbol{y}) &\leftarrow \mathsf{Proj}^{i}(x, \boldsymbol{y}, 1), A_i(x)\\ \mathsf{Proj}^{i-1}(x, \boldsymbol{y}) &\leftarrow \mathsf{Proj}^{i}(x, \boldsymbol{y}, 0), \overline{\mathsf{A}}_i(x)\end{aligned}$$

Intuitively, $\mathsf{Proj}^{i-1}(a, b_1, \ldots, b_{i-1})$ says the partial type given by the bit values $b_1, \ldots, b_{i-1}$ can be extended to a marked type by choosing additional concepts according to the actual type of the individual a. Thus the fact $\mathsf{Proj}^0(a)$ represent the situation where a has a marked type. Such situations are ruled out by adding the constraint:

$$\leftarrow \mathsf{Proj}^0(x)$$

This concludes the translation of the instance query $(\mathcal{T}, q)$, where $\mathcal{T}$ is an $\mathcal{ALCHI}$ TBox, into the disjunctive Datalog program P. This construction, together with Theorems 1 and 2, yields the main result of this paper.

Theorem 3. *For an instance query $(\mathcal{T}, q)$, we can build in polynomial time a* Datalog *query (P, q) such that the certain answers to $(\mathcal{T}, q)$ and (P, q) coincide for any given ABox $\mathcal{A}$ over the signature of $\mathcal{T}$.*

The above encoding employs disjunctive programs. Entailment of ground atoms in such programs is coNExpTime-complete [7], which does not match the ExpTime-completeness of satisfiability of $\mathcal{ALCHI}$ KBs. However, we employ disjunction in a limited way, and thus our programs fall into a class of programs that can be evaluated in (deterministic) exponential time. In particular, the above program P can be partitioned into two programs P_1, P_2 as follows: P_1 consists of all rules in (I) and (II), and P_2 consists of the remaining rules. Observe that P_1 is a disjunctive program with at most two variables in each rule, while P_2 is is disjunction-free. Note also that P_2 does not define any relations used in P_1, i.e. none of the relation symbols of P_1 occurs in the head of a rule in P_2. Assume a set F of facts over the signature of P_1. Due to the above properties, the successful runs of the following non-deterministic procedure generate the set of all minimal models of $P \cup F$:

(S1) Compute a minimal model I_1 of $P_1 \cup F$. If I_1 does not exist due to a constraint violation, then return *failure*.
(S2) Compute the least model I_2 of $P_2 \cup I_1$. Again, if I_2 does not exist due to a constraint violation, then return *failure*. Otherwise, output I_2.

Since P_1 has at most two-variables in every rule, each minimal model I_1 of $P_1 \cup F$ is of polynomial size in the size of $P_1 \cup F$, and the set of all such models can be traversed in polynomial space. For a given I_1, performing the step (S2) is feasible in (deterministic) exponential time, because P_2 is disjunction-free and of polynomial size in the size of the original instance query. It follows that computing the certain answers to (P, q) for any given ABox $\mathcal{A}$ over the signature of $\mathcal{T}$ requires single exponential time.

5 Discussion

We have presented our results for $\mathcal{ALCHI}$, but they also apply to $\mathcal{SHI}$, its extension with transitivity axioms; it is well known that instance queries mediated by $\mathcal{SHI}$ TBoxes can be rewritten in polynomial time to instance queries mediated by $\mathcal{ALCHI}$ TBoxes (see, e.g., [15]). Moreover, we have presented the translation for *instance queries*, but the results can be easily generalized, e.g., to DL-safe rules of [22], or the *quantifier-free conjunctive queries* like in [21]. These queries are syntactically restricted to ensure that the relevant variable assignments only map into individuals of the input ABox. We note that generalizing our translation to arbitrary conjunctive queries while remaining polynomial is not possible under common assumptions in complexity theory. Indeed, cautious inference from a disjunctive program is in coNExpTime, but entailment of (Boolean) conjunctive queries is 2ExpTime-hard already for the DL $\mathcal{ALCI}$ [19].

As mentioned in the introduction, both the game theoretic characterization and the Datalog translation can be extended to ontologies in $\mathcal{ALCHOI}$, the extension of $\mathcal{ALCHI}$ with nominals. This is possible even in the presence of the so-called *closed predicates*, which are useful for combining complete and incomplete knowledge, by explicitly specifying predicates assumed complete, thus given a *closed-world* semantics [10, 20]. For example, take the following TBox $\mathcal{T}$:

$$\mathsf{BScStud} \sqsubseteq \mathsf{Student} \qquad \mathsf{Student} \sqsubseteq \exists \mathsf{attends}.\mathsf{Course}$$
$$\mathsf{BScStud} \sqsubseteq \forall \mathsf{attends}.\neg \mathsf{GradCourse}$$

and the ABox $\mathcal{A}$:

$$\mathsf{Course}(c_1),\ \mathsf{Course}(c_2),\ \mathsf{GradCourse}(c_2),\ \mathsf{BScStud}(a)$$

Then (a, c_1) is not a certain answer to $(\mathcal{T}, q)$ with $q = \mathsf{attends}(x, y)$. But if c_1 and c_2 are known to be the *only* courses, then (a, c_1) should be a certain answer. This can be achieved by declaring Course a *closed predicate*: (a, c_1) is indeed a certain answer to the *OMQ with closed predicates* $(\mathcal{T}, \Sigma, q)$, where $\Sigma = \{\mathsf{Course}\}$.

Interestingly, OMQs with closed predicates are *non-monotonic*. Indeed, (a, c_1) is a certain answer to $(\mathcal{T}, \Sigma, q)$ over $\mathcal{A}$, but it is not a certain answer over the extended set of facts $\mathcal{A}' = \mathcal{A} \cup \{\mathsf{Course}(c_3)\}$. For this reason, these queries cannot be rewritten into monotonic variants of Datalog, like positive Datalog with disjunction as considered here. However, one can devise a polynomial time translation into *disjunctive* Datalog *with negation as failure*, for OMQs of the form $(\mathcal{T}, \Sigma, q)$ where $\mathcal{T}$ is an $\mathcal{ALCHOI}$ TBox, Σ is a set of closed predicates, and q is an instance query. The translation uses the same ideas presented here, but handling closed predicates (both in the game-theoretic characterization, and in the Datalog translation) implies reasoning about the types that are already realized in the core; in Datalog, this requires negation. Details can be found in [1].

Acknowledgements

This work was supported by the Austrian Science Fund (FWF): projects P25207-N23, T515 and W1255-N23.

References

1. S. Ahmetaj, M. Ortiz, and M. Šimkus. Polynomial datalog rewritings for expressive description logics with closed predicates. In *Proc. of IJCAI'16*, 2016. To appear.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, second edition, 2007.
3. M. Bienvenu and M. Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Proc. of RW 2015*, pages 218–307. Springer, 2015.
4. M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
6. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
7. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.
8. T. Eiter, M. Ortiz, and M. Šimkus. Conjunctive query answering in the description logic SH using knots. *J. Comput. Syst. Sci.*, 78(1):47–85, 2012.
9. T. Eiter, M. Ortiz, M. Šimkus, T. Tran, and G. Xiao. Query rewriting for Horn-SHIQ plus rules. In *Proc. of AAAI 2012*. AAAI Press, 2012.
10. E. Franconi, Y. A. Ibáñez-García, and I. Seylan. Query answering with DBoxes is hard. *Electr. Notes Theor. Comput. Sci.*, 278:71–84, 2011.
11. G. Gottlob, S. Kikot, R. Kontchakov, V. V. Podolskii, T. Schwentick, and M. Zakharyaschev. The price of query rewriting in ontology-based data access. *Artif. Intell.*, 213:42–59, 2014.
12. G. Gottlob, M. Manna, and A. Pieris. Polynomial combined rewritings for existential rules. In *Proc. of KR 2014*. AAAI Press, 2014.
13. G. Gottlob, M. Manna, and A. Pieris. Polynomial rewritings for linear existential rules. In *Proc. of IJCAI 2015*. AAAI Press, 2015.
14. G. Gottlob and T. Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In *Proc. of KR 2012*. AAAI Press, 2012.
15. U. Hustadt, B. Motik, and U. Sattler. Reasoning in description logics by a reduction to disjunctive datalog. *J. Autom. Reasoning*, 39(3):351–384, 2007.
16. M. Kaminski, Y. Nenov, and B. C. Grau. Computing datalog rewritings for disjunctive datalog programs and description logic ontologies. In *Proc. of RR 2014*, pages 76–91, 2014.
17. M. Kaminski, Y. Nenov, and B. C. Grau. Datalog rewritability of disjunctive datalog programs and its applications to ontology reasoning. In *Proc. of AAAI 2014*, pages 1077–1083, 2014.
18. R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to ontology-based data access. In *Proc. of IJCAI 2011*. IJCAI/AAAI, 2011.
19. C. Lutz. Inverse roles make conjunctive queries hard. In *Proc. of DL 2007*. CEUR-WS.org, 2007.
20. C. Lutz, I. Seylan, and F. Wolter. Ontology-based data access with closed predicates is inherently intractable(sometimes). In *Proc. of IJCAI 2013*. IJCAI/AAAI, 2013.

21. C. Lutz, I. Seylan, and F. Wolter. Ontology-mediated queries with closed predicates. In *Proc. of IJCAI 2015*. IJCAI/AAAI, 2015.
22. B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *J. Web Sem.*, 3(1):41–60, 2005.
23. M. Ortiz, S. Rudolph, and M. Šimkus. Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In *Proc. of KR 2010*. AAAI Press, 2010.
24. H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable query answering and rewriting under description logic constraints. *J. Applied Logic*, 8(2):186–209, 2010.
25. D. Trivela, G. Stoilos, A. Chortaras, and G. B. Stamou. Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Sem.*, 33:30–49, 2015.