Polynomial Datalog Rewritings for Expressive Description Logics with Closed Predicates

Shqiponja Ahmetaj, Magdalena Ortiz, and **Mantas Šimkus**
Institute of Information Systems,
TU Vienna, Austria

Abstract

We consider instance queries mediated by an ontology expressed in the expressive DL $\mathcal{ALCHIO}$ with closed predicates. We observe that such queries are non-monotonic and cannot be expressed in monotonic variants of DATALOG, but a polynomial time translation into disjunctive DATALOG extended with negation as failure is feasible. If no closed predicates are present–in the case of classical instance checking in $\mathcal{ALCHIO}$–our translation yields a positive disjunctive DATALOG program of polynomial size. To the best of our knowledge, this is the first polynomial time translation of an expressive (non-Horn) DL into disjunctive DATALOG.

1 Introduction

In *ontology-mediated queries* (OMQs), a database query is enriched with an ontology, providing domain 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$\pm$, see e.g., [Bienvenu *et al.*, 2014; Bienvenu and Ortiz, 2015; Gottlob *et al.*, 2015] and their references.

The *open-world semantics* of these formalisms makes them suitable for handling incomplete knowledge, but viewing *all* data as incomplete can result in too few *certain answers*. For this reason, *closed predicates* have been advocated as a powerful tool to combine complete and incomplete knowledge, by explicitly specifying predicates assumed complete, thus given a *closed-world* semantics [Franconi *et al.*, 2011; Lutz *et al.*, 2013]. For example, take the following ontology $\mathcal{T}$ (formally, a DL *TBox*):

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

and the following set of facts $\mathcal{A}$ (an *ABox*):

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

Then (a, c_1) is not a certain answer to the *instance query* $q = \mathsf{attends}(x, y)$ mediated by $\mathcal{T}$, but if c_1 and c_2 are known to be the *only* courses, then (a, c_1) should become a certain answer. This can be achieved by declaring Course a *closed predicate*.

In this paper, we investigate the relative expressiveness of OMQs in terms of more traditional query languages like DATALOG. More precisely, we are interested in the following problem: given an OMQ Q (specified by a query and a TBox, possibly with closed predicates), obtain a DATALOG query Q'—in a suitable fragment—such that, for any ABox $\mathcal{A}$, the certain answers to Q and Q' coincide.

The existence of such a Q' and its size are crucial for understanding the expressive power and succinctness of different families of OMQs. However, they are also very relevant in practice, since they allow to reuse existing database technologies to support OMQ answering. For example, the research into OMQs that can be rewritten into *first-order (FO) queries* has produced the successful *DL-Lite* family [Calvanese *et al.*, 2007]. The succinctness of FO-rewritings for DL-Lite, and for families of existential rules that are FO-rewritable, has been extensively studied [Gottlob *et al.*, 2014a; Gottlob and Schwentick, 2012], and for cases where (succinct) FO-rewritings do not exists, some authors have considered rewritings that, unlike the ones we consider here, are not *data-independent* [Kontchakov *et al.*, 2011; Gottlob *et al.*, 2014b].

In the presence of closed predicates, the only rewritability results are FO-rewritability for the core fragment of DL-Lite [Lutz *et al.*, 2015], and a rewriting algorithm for queries that satisfy some strong *definability* criteria [Seylan *et al.*, 2009]. Other works on OMQs with closed predicate have focused on the complexity of their evaluation, e.g., [Ngo *et al.*, 2015; Lutz *et al.*, 2013; Franconi *et al.*, 2011]. The latter two have shown coNP-hardness in data complexity for many lightweight DLs, barring the existence of FO-rewritings.

Many DLs are not FO-rewritable, but can be rewritten into monotonic DATALOG queries, leading to implemented systems, e.g., [Pérez-Urbina *et al.*, 2010; Eiter *et al.*, 2012b; Trivela *et al.*, 2015]. The pioneering work in [Hustadt *et al.*, 2007] 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 *conjunctive queries (CQs)* in expressive DLs without closed predicates ($\mathcal{SH}$, $\mathcal{SHQ}$) to programs in disjunctive DATALOG was

introduced in [Eiter *et al.*, 2012a], but the program may contain double exponentially many predicates. For $\mathcal{ALC}$ and for union of CQs, the existence of exponential rewritings into disjunctive DATALOG was shown recently [Bienvenu *et al.*, 2014], and for restricted fragments of $\mathcal{SHI}$ and classes of CQs translations to DATALOG were investigated in [Kaminski *et al.*, 2014a; 2014b]. A polynomial time DATALOG translation of instance queries was proposed in [Ortiz *et al.*, 2010], 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.

In this paper, we consider the class $\mathcal{Q}$ of OMQs of the form $(\mathcal{T}, \Sigma, q)$, where q is an instance query and $\mathcal{T}$ is a TBox in the very expressive DL $\mathcal{ALCHIO}$ with closed predicates Σ. We observe that such queries are *non-monotonic*. Indeed, if we take $\Sigma = \{\mathsf{Course}\}$ as the set of closed predicates in the above example, then (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 or without disjunction). The main contribution of this paper is a polynomial time translation of queries in $\mathcal{Q}$ into *disjunctive* DATALOG *extended with negation as failure*. Our translation is modular: if no closed predicates are present—in the case of classical instance queries in $\mathcal{ALCHIO}$—our translation yields a positive disjunctive DATALOG program of polynomial size. A simplified version of this translation for $\mathcal{ALCHI}$ can be found in [Ahmetaj *et al.*, 2016]. To our knowledge, this is the first polynomial time translation of an expressive (non-Horn) DL into disjunctive DATALOG.

2 Preliminaries

We give some basic notions of DLs and DATALOG.

The DL $\mathcal{ALCHIO}$ 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 as follows:

(i) $\top$, $\bot$ and all $A \in \mathsf{N_C}$ are concepts;

(ii) if $o \in \mathsf{N_I}$, then $\{o\}$ is a concept;

(iii) if C_1, C_2 are concepts, then $C_1 \sqcap C_2$, $C_1 \sqcup C_2$ and $\neg C_1$ are concepts;

(iv) 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 $A(a)$ (called *concept assertion*) and $p(a, b)$ (called *role assertion*), where $\{a, b\} \subseteq \mathsf{N_I}$, $A \in \mathsf{N_C}$, and $p \in \mathsf{N_R}$. A *knowledge base (KB) (with closed predicates)* is a triple $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, where $\mathcal{T}$ is a TBox, $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ is the set of *closed predicates* in $\mathcal{K}$, 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 [Baader *et al.*, 2007].

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 $q(\vec{a})$ if: $(\vec{a})^{\mathcal{I}} \in q^{\mathcal{I}}$, in symbols, $\mathcal{I} \models q(\vec{a})$. For Γ a TBox or ABox, we write $\mathcal{I} \models \Gamma$ if $\mathcal{I} \models \alpha$ for all $\alpha \in \Gamma$. For an ABox $\mathcal{A}$ and $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$, we write $\mathcal{I} \models_\Sigma \mathcal{A}$ if

(a) $\mathcal{I} \models \mathcal{A}$,

(b) for all $A \in \Sigma \cap \mathsf{N_C}$, if $e \in A^{\mathcal{I}}$, then $A(e) \in \mathcal{A}$, and

(c) for all $r \in \Sigma \cap \mathsf{N_R}$, if $(e_1, e_2) \in r^{\mathcal{I}}$, then $r(e_1, e_2) \in \mathcal{A}$.

For a KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, we write $\mathcal{I} \models \mathcal{K}$ if the following hold:[1] *(i)* $a \in \Delta^{\mathcal{I}}$ and $a^{\mathcal{I}} = a$ for each $a \in \mathsf{N_I}$ occurring in $\mathcal{K}$, *(ii)* $\mathcal{I} \models \mathcal{T}$, and *(iii)* $\mathcal{I} \models_\Sigma \mathcal{A}$. For an assertion α, we write $\mathcal{K} \models \alpha$ if $\mathcal{I} \models \alpha$ for all $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$.

Instance Queries In this paper we consider an *(ontology mediated) instance query*. Each such query is given as a triple $Q = (\mathcal{T}, \Sigma, q)$, where $\mathcal{T}$ is a TBox, $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$, and $q \in \mathsf{N_C} \cup \mathsf{N_R}$. Let $\vec{a} \in \mathsf{N_I}$ in case $q \in \mathsf{N_C}$, and $\vec{a} \in \mathsf{N_I}^2$ otherwise. Then $\vec{a}$ is a *certain answer* to Q over an ABox $\mathcal{A}$ if $(\mathcal{T}, \Sigma, \mathcal{A}) \models q(\vec{a})$; note that if $\Sigma = \emptyset$, this boils down to the usual DL instance checking problem. To ease presentation, we assume that every concept and role of $\mathcal{A}$ also occurs in $\mathcal{T}$.

Normal Form We let $\mathsf{N_C^+} = \mathsf{N_C} \cup \{\{a\} \mid a \in \mathsf{N_I}\} \cup \{\top, \bot\}$ be the *basic concepts*. Our results apply to arbitrary TBoxes, but to simplify presentation, we consider TBoxes in *normal form* where inclusions take one of the following forms:

$$(\mathbf{N1})\ B_1 \sqcap \cdots \sqcap B_n \sqsubseteq B_{n+1} \sqcup \cdots \sqcup B_k$$

$$(\mathbf{N2})\ A \sqsubseteq \exists r.A' \qquad (\mathbf{N3})\ A \sqsubseteq \forall r.A' \qquad (\mathbf{N4})\ r \sqsubseteq s$$

with r, s roles, $\{B_1, \ldots, B_k\} \subseteq \mathsf{N_C^+}$, 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 $\mathsf{N_I}(\Gamma)$, $\mathsf{N_R}(\Gamma)$, $\mathsf{N_C}(\Gamma)$, and $\mathsf{N_C^+}(\Gamma)$ the set of individuals, role names, concept names, and basic concepts that occur in Γ, resp. We write $r^{(-)} \in \Sigma$ to mean $r \in \Sigma$ if $r \in \mathsf{N_R}$, and $r^- \in \Sigma$ otherwise.

Nonmonotonic 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 $R(t_1, \ldots, t_n)$, where $\{t_1, \ldots, t_n\} \subseteq \mathsf{N_I} \cup \mathsf{N_V}$, and R is an n-ary relation symbol. A *negated atom* is an expression of the form $not\ \alpha$, where α is an atom. A *rule* ρ is an expression of the form $h_1 \vee \ldots \vee h_n \leftarrow b_1, \ldots, b_k$, where $H = \{h_1, \ldots, h_n\}$ is a set of atoms, called the *head* of ρ, and $B = \{b_1, \ldots, b_k\}$ is a set of possibly negated atoms, called the *body* of ρ. Each variable that occurs in the head of ρ must also occur in a (non-negated) atom in the body of ρ. Rules

[1] We make the *standard name assumption (SNA)*, which is common for closed predicates, for the individuals occurring in $\mathcal{K}$.

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. If negated atoms do not occur in P, then P is *positive*.

An *(Herbrand) interpretation* (or, *database*) I is any finite set of variable-free (or, *ground*) atoms. We assume a binary built-in *inequality* predicate $\neq$ with a natural meaning: in any interpretation I, $a \neq b \in I$ iff (i) $a \neq b$, and (ii) a (resp. b) occurs in an atom $R(\vec{t}) \in I$, where R is not $\neq$. An interpretation I is a *model* of a positive program P if $\{b_1, \ldots, b_k\} \subseteq I$ implies $I \cap \{h_1, \ldots, h_n\} \neq \emptyset$ for all rules $h_1 \vee \ldots \vee h_n \leftarrow b_1, \ldots, b_k$ in $\mathsf{ground}(P)$. We say an interpretation I is a *minimal model* of a positive program P if I is a model of P, and there is no $J \subsetneq I$ that is a model of P.

The *GL-reduct* of a program P w.r.t. an interpretation I is the program P^I that is obtained from $\mathsf{ground}(P)$ in two steps: (i) deleting every rule that has $not\ \alpha$ in the body with $\alpha \in I$, and (ii) deleting all negated atoms in the remaining rules [Gelfond and Lifschitz, 1988]. An interpretation I is a *stable model* (also known as *answer sets*) of a program P if I is a minimal model of P^I.

A *query* is a pair (P, q), where P is a program, and q is a predicate symbol from P. A tuple $\vec{a}$ of constants is a *certain answer to* (P, q) *over a database* I if $q(\vec{a}) \in J$ for all stable models J of $P \cup I$.

3 Characterization of Counter Models

Assume a KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ and an assertion q. Towards deciding $\mathcal{K} \not\models q$ 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 $\mathsf{N_I}(\mathcal{K})$. Core interpretations fix how the individuals of $\mathcal{K}$ participate in concepts and roles, ensuring the satisfaction of $\mathcal{A}$ and Σ, and the non-entailment of q.

(2) Check that $\mathcal{I}_c$ can be extended to satisfy all axioms in $\mathcal{T}$.

Defining 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 the notion of core interpretations.

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

(c1) $\Delta^{\mathcal{I}_c} = \mathsf{N_I}(\mathcal{K})$ *and* $a^{\mathcal{I}_c} = a$ *for all* $a \in \mathsf{N_I}(\mathcal{K})$,

(c2) $\mathcal{I}_c \models_\Sigma \mathcal{A}$,

(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) $\mathcal{I}_c \models A \sqsubseteq \exists r.A'$ *for all* $A \sqsubseteq \exists r.A' \in \mathcal{T}$, *if* $r \sqsubseteq s \in \mathcal{T}$ *for some* $s^{(-)} \in \Sigma$.

(c6) $q^{\mathcal{I}_c} = \emptyset$ *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* $\mathsf{N_I}(\mathcal{K})$, *and* $q^{\mathcal{J}} = q^{\mathcal{I}_c}$ *for all* $q \in \Sigma$.

A core and its extensions coincide on the assertions they entail, and deciding non-entailment of an instance query amounts to deciding whether there is a core that does not entail it, and that can be extended into a model. But verifying whether a core can be extended into a full model is hard, as 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 $\mathcal{ALCHOI}$. 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 nonmonotonic disjunctive DATALOG. For this we use *types*, which we define as follows:

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* $\alpha = B_1 \sqcap \cdots \sqcap B_n \sqsubseteq B_{n+1} \sqcup \cdots \sqcup B_k$ *of type* (**N1**), *if* $\{B_1, \ldots, B_n\} \subseteq \tau$ *implies* $\{B_{n+1}, \ldots, B_k\} \cap \tau \neq \emptyset$; *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 *type*$(a, \mathcal{I}_c)$. If it doesn't satisfy certain local consistency conditions (e.g., the inclusions (**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.

Formally, for a TBox $\mathcal{T}$, a set $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ and an interpretation $\mathcal{I}$, we define the *locally consistent* set $\mathsf{LC}(\mathcal{T}, \Sigma, \mathcal{I})$ as the set of types τ over $\mathcal{T}$ such that:

(LC$_{\mathrm{N1}}$*)* τ satisfies all inclusions of type (**N1**) in $\mathcal{T}$.

(LC$_\Sigma$*)* τ must be realized in $\mathcal{I}$ if one of the following holds:

(i) $\tau \cap \Sigma \neq \emptyset$, or

(ii) $A \in \tau$, $A \sqsubseteq \exists r.A' \in \mathcal{T}$, $r \sqsubseteq s \in \mathcal{T}$ and $s^{(-)} \in \Sigma$.

(LC$_\mathcal{O}$*)* If τ contains a nominal $\{a\}$, then $\tau =$ *type*$(a, \mathcal{I})$.

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

($\diamond$) If $\tau \notin \mathsf{LC}(\mathcal{T}, \Sigma, \mathcal{I})$, then Sam is declared 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 wins the game. 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 $\diamond$.

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 inclusions 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*, $\mathit{tail}(w) \in \mathsf{LC}(\mathcal{T},\Sigma,\mathcal{I})$ 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 q. Then $\mathcal{K} \not\models q$ iff there is a core interpretations $\mathcal{I}_c$ for $\mathcal{K}$ such that:*

(1) $\mathcal{I}_c \not\models q$, 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 q.

For the "$\Leftarrow$" direction, we assume an arbitrary extension $\mathcal{J}$ of $\mathcal{I}_c$ that models $\mathcal{K}$, and let T be a the set of all the types realized in $\mathcal{J}$. For each $\tau \in T$ and each $A \sqsubseteq \exists r.A' \in \mathcal{T}$ with $A \in \tau$, let $\mathit{str}(\tau, A \sqsubseteq \exists r.A') = \tau'$ for an arbitrarily chosen $\tau' \in T$ that satisfies (C1) and (C2), which exists because $\mathcal{J}$ is a model. This *str* is a non-losing strategy for Bob on $\mathcal{I}_c$.

For the "$\Rightarrow$" direction, from an arbitrary non-losing *str* for $\mathcal{I}_c$, we build $\mathcal{J}$ as follows. We call a type τ *fixed* if it contains a nominal $\{a\}$, some $A \in \Sigma$, or some A with $A \sqsubseteq \exists r.A' \in \mathcal{T}$, $r \sqsubseteq s \in \mathcal{T}$ and $s^{(-)} \in \Sigma$; note that fixed types cannot be realized by domain objects outside the core. For each fixed τ that is realized in $\mathcal{I}_c$, we let a_τ denote a fixed, arbitrary individual realizing τ (in particular, $a_\tau = b$ if $\{b\} \in \tau$).

We denote by $\mathit{frn}(\mathcal{I}_c, \mathit{str})$ the set of all finite runs $a\alpha_1\tau_1\cdots$ that follow *str* where each τ_i, $i \geq 1$, are *not* fixed types, and by $\mathit{fxr}(\mathcal{I}_c, \mathit{str})$ the set of all finite runs $a\alpha_1\tau_1\cdots\alpha_\ell\tau_\ell$ that follow *str*, where τ_ℓ is a fixed type, and each τ_i, $1 \leq i < \ell$, is not a fixed type. The domain of $\mathcal{J}$ is:

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

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

$$a^{\mathcal{J}} = a^{\mathcal{I}_c} \qquad A^{\mathcal{J}} = \{w \mid A \in \mathit{tail}(w)\}$$

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

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

Note that the interpretation is well defined, since $w\alpha_i\tau_i \in \mathit{fxr}(\mathcal{I}_c, \mathit{str})$ implies that τ_i is a fixed type in $\mathsf{LC}(\mathcal{T},\Sigma,\mathcal{I}_c)$, which guarantees that τ_i is realized in $\mathcal{I}_c$ and that a_{τ_i} exists. One can show that $\mathcal{J} \models \mathcal{K}$. Roughly, $\mathcal{J} \models_\Sigma \mathcal{A}$ is a direct consequence of the definition of $\mathcal{I}_c$ and the fact that $\mathcal{J}$ is its extension. We can similarly argue that $\mathcal{J}$ satisfies the inclusions in $\mathcal{T}$ for the individuals. For the fresh objects, it follows from the fact that they have as children in $\mathcal{J}$ the types given by the strategy, which are suitable successors. □

Algorithm 1: Mark

input : TBox $\mathcal{T}$, set $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$, interpretation $\mathcal{I}$

output: Set of (possibly marked) types

$N \leftarrow \{\tau \mid \tau \text{ is a type over } \mathcal{T}\}$

($\mathrm{M_{N1}}$) Mark each $\tau \in N$ that violates some inclusion of the form $(\mathbf{N1})$ in $\mathcal{T}$

(M_Σ) Mark each $\tau \in N$ such that τ is not realized in $\mathcal{I}$ and one of the following holds:

(i) $\tau \cap \Sigma \neq \emptyset$, or

(ii) $A \in \tau$, $A \sqsubseteq \exists r.A' \in \mathcal{T}$, $r \sqsubseteq s \in \mathcal{T}$, $s^{(-)} \in \Sigma$

($\mathrm{M}_\mathcal{O}$) Mark each $\tau \in N$ with $\{a\} \in \tau$ and $\tau \neq \mathit{type}(a,\mathcal{I})$

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) τ' is marked,

(C1′) $A' \notin \tau'$, or

(C2′) there exists $A_1 \sqsubseteq \forall s.A_2 \in \mathcal{T}$ with

(i) $r \sqsubseteq s \in \mathcal{T}$ and $A_1 \in \tau$ and $A_2 \notin \tau'$, or

(ii) $r^- \sqsubseteq s \in \mathcal{T}$ and $A_1 \in \tau'$ and $A_2 \notin \tau$

until *no new type is marked*

return N

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 strategy to defeat Bob. It takes as input the TBox $\mathcal{T}$, the set $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$, 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 marks types that are bad choices for Bob (since they give Sam a way to defeat him). In steps ($\mathrm{M_{N1}}$), (M_Σ) and ($\mathrm{M}_\mathcal{O}$) the algorithm respectively marks in N all types that violate the condition ($\mathrm{LC_{N1}}$), (LC_Σ) or ($\mathrm{LC}_\mathcal{O}$); 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 which Bob cannot reply with any τ'.

The formal relationship between the game and the marking algorithm is established in the following theorem.

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 $\mathbf{Mark}(\mathcal{T},\Sigma,\mathcal{I}_c)$.*

Proof. (Sketch.) For the "$\Rightarrow$" direction, we can show (by induction in the number of iterations of $\mathbf{Mark}(\mathcal{T},\Sigma,\mathcal{I}_c)$) 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). $\square$

4 Rewriting into Nonmonotonic Disjunctive Datalog

Assume an instance query $(\mathcal{T}, \Sigma, q)$. We build next a polynomially sized program P such that the queries $(\mathcal{T}, \Sigma, q)$ and (P, q) have the same certain answers for all ABoxes over the signature of $\mathcal{T}$. Roughly, the program P consists of 3 major components described as follows:

(a) rules to non-deterministically generate a core interpretation $\mathcal{I}_c$ for the KB $(\mathcal{T}, \Sigma, \mathcal{A})$, 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 $\mathcal{T}$ or the input ABox. For each $A \in \mathsf{N_C}(\mathcal{T})$, $r \in \mathsf{N_R}(\mathcal{T})$, and nominal $\{a\}$ that occurs in $\mathcal{T}$ we have:

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

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

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

To ensure *(c3)* in Definition 1, for each $A \sqsubseteq \forall r.A' \in \mathcal{T}$ we add one of the following:

$$\begin{aligned} &\leftarrow A(x), r(x,y), not\ A'(y) && \text{if } A' \in \Sigma \\ A'(y) &\leftarrow A(x), r(x,y) && \text{if } A' \notin \Sigma \end{aligned}$$

In principle, a constraint $\leftarrow A(x), r(x,y), not\ A'(y)$ could be for all concept names, not only closed ones. We make the distinction here and later in order to eventually obtain a positive program in case $\Sigma = \emptyset$.

To ensure *(c4)*, for each role inclusion $r \sqsubseteq s \in \mathcal{T}$ we add one of the following:

$$\begin{aligned} &\leftarrow r(x,y), not\ s(x,y) && \text{if } s^{(-)} \in \Sigma \\ s(x,y) &\leftarrow r(x,y) && \text{if } s^{(-)} \notin \Sigma \end{aligned}$$

To ensure *(c5)*, for each inclusion $A \sqsubseteq \exists r.A' \in \mathcal{T}$ with $r \sqsubseteq s \in \mathcal{T}$ and $s^{(-)} \in \Sigma$, add the following rules

$$\begin{aligned} R_{A'}(x) &\leftarrow r(x,y), A'(y) \\ &\leftarrow A(x), not\ R_{A'}(x) \end{aligned}$$

Here $R_{A'}$ is a fresh predicate symbol.

Intuitively, the stable models of the above rules generate the different core interpretations $\mathcal{I}_c$ of the KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ 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 and nominals in $\mathcal{T}$. Assume an arbitrary enumeration $B_1, \ldots, B_k$ of $\mathsf{N_C^+}(\mathcal{T})$, i.e. of the concept names and nominals that occur in $\mathcal{T}$. 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 $(b_1, \ldots, b_k) \in \mathsf{Type}$ encodes the type $\tau = \{B_i \mid b_i = 1, 1 \le i \le k\} \cup \{\top\}$. 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 We use k-ary relation symbols first and last, and add the facts $\mathsf{first}(0, \ldots, 0) \leftarrow$ and $\mathsf{last}(1, \ldots, 1) \leftarrow$. We also use a $2k$-ary relation next that will store a lexicographic ordering over $\{0,1\}^k$ with $(0, \ldots, 0)$ and $(1, \ldots, 1)$ the first and the last element, respectively. That is, given $\vec{u}, \vec{v} \in \{0,1\}^k$, the fact $\mathsf{next}(\vec{u}, \vec{v})$ is true if $\vec{v}$ follows $\vec{u}$ in a lexicographic ordering. The rules to populate next are standard (see, e.g., Theorem 4.5 in [Dantsin *et al.*, 2001]).

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

$$\begin{aligned} \mathsf{Type}(\vec{x}) &\leftarrow \mathsf{first}(\vec{x}) \\ \mathsf{Type}(\vec{y}) &\leftarrow \mathsf{next}(\vec{x}, \vec{y}) \end{aligned}$$

(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 $\vec{x}$, we let $B \in \vec{x}$ 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 $B \notin \vec{x}$ denote the atom $F(x_j)$, where j is the index of B in the enumeration.

Then the step ($\mathrm{M_{N1}}$), which marks types violating inclusions of type (**N1**), is implemented using the following rule for every inclusion $B_1 \sqcap \cdots \sqcap B_n \sqsubseteq B_{n+1} \sqcup \cdots \sqcup B_k \in \mathcal{T}$:

$$\begin{aligned} \mathsf{Marked}(\vec{x}) \leftarrow &\mathsf{Type}(\vec{x}), B_1 \in \vec{x}, \ldots, B_n \in \vec{x}, \\ &B_{n+1} \notin \vec{x}, \ldots, B_k \notin \vec{x} \end{aligned}$$

(V) Collecting realized types We employ $(i+1)$-ary ("has type") relations hasT^i for all $0 \le i \le k$. We add the rule $\mathsf{hasT}^0(x) \leftarrow \mathsf{ind}(x)$, and the following rules for all $1 \le i \le k$:

$$\begin{aligned} \mathsf{hasT}^i(x, \vec{y}, 1) &\leftarrow \mathsf{hasT}^{i-1}(x, \vec{y}), B_i(x) && B_i \in \mathsf{N_C} \\ \mathsf{hasT}^i(a, \vec{y}, 1) &\leftarrow \mathsf{hasT}^{i-1}(a, \vec{y}) && B_i = \{a\} \\ \mathsf{hasT}^i(x, \vec{y}, 0) &\leftarrow \mathsf{hasT}^{i-1}(x, \vec{y}), x \neq a && B_i = \{a\} \\ \mathsf{hasT}^i(x, \vec{y}, 0) &\leftarrow \mathsf{hasT}^{i-1}(x, \vec{y}), \overline{B}_i(x) && B_i \in \mathsf{N_C} \setminus \Sigma \\ \mathsf{hasT}^i(x, \vec{y}, 0) &\leftarrow \mathsf{hasT}^{i-1}(x, \vec{y}), not\ B_i(x) && B_i \in \mathsf{N_C} \cap \Sigma \end{aligned}$$

Intuitively, hasT^k stores the type of individuals in a generated core interpretation for the KB. I.e. the fact $\mathsf{hasT}^k(c, \vec{v})$ says that the individual c has the type that is encoded in the bit

vector $\vec{v}$. We can now project away the individuals and store in the relation RealizedType the set of realized types:

$$\mathsf{RealizedType}(\vec{y}) \leftarrow \mathsf{hasT}^k(x, \vec{y})$$

(VI) Implementing Step ($\mathbf{M}_\Sigma$) In this step, we mark all non-realized types that contain a closed concept name, or enforce an s-neighbor with $s^{(-)} \in \Sigma$. In particular, for (i) every $A \in \mathsf{N_C}(\mathcal{T}) \cap \Sigma$, and (ii) every inclusion $A \sqsubseteq \exists r.A' \in \mathcal{T}$ with $r \sqsubseteq s \in \mathcal{T}$ and $s^{(-)} \in \Sigma$, we add the rule

$$\mathsf{Marked}(\vec{x}) \leftarrow \mathsf{Type}(\vec{x}), A \in \vec{x}, \mathit{not}\ \mathsf{RealizedType}(\vec{x})$$

(VII) Implementing Step ($\mathbf{M}_\mathcal{O}$) This step marks every nominal-containing type that is not realized in the core interpretation. We implement it by adding the following rules for all $B \in \mathsf{N_C^+}$ and all nominals $\{a\}$ that occur in $\mathcal{T}$:

$$\mathsf{Marked}(\vec{x}) \leftarrow \mathsf{Type}(\vec{x}), \{a\} \in \vec{x}, \mathsf{hasT}^k(a, \vec{y}), B \in \vec{x}, B \notin \vec{y}$$

$$\mathsf{Marked}(\vec{x}) \leftarrow \mathsf{Type}(\vec{x}), \{a\} \in \vec{x}, \mathsf{hasT}^k(a, \vec{y}), B \notin \vec{x}, B \in \vec{y}$$

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

- For collecting each τ' that satisfies (C0), we add:

$$\mathsf{MarkedOne}_\alpha(\vec{x}, \vec{y}) \leftarrow \mathsf{Type}(\vec{x}), \mathsf{Marked}(\vec{y})$$

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

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

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

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

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

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

Intuitively, we want to infer $\mathsf{Marked}(\vec{t})$ if A is set to true in $\vec{t}$ and $\mathsf{MarkedOne}_\alpha(\vec{t}, \vec{v})$ is true for all types (bit vectors) $\vec{v}$. To achieve this, we rely on another auxiliary $2k$-ary relation $\mathsf{MarkedUntil}_\alpha$ for each inclusion α:

$$\mathsf{MarkedUntil}_\alpha(\vec{x}, \vec{z}) \leftarrow \mathsf{MarkedOne}_\alpha(\vec{x}, \vec{z}), \mathsf{first}(\vec{z})$$

$$\mathsf{MarkedUntil}_\alpha(\vec{x}, \vec{u}) \leftarrow \mathsf{MarkedUntil}_\alpha(\vec{x}, \vec{z}), \mathsf{next}(\vec{z}, \vec{u}), \mathsf{MarkedOne}_\alpha(\vec{x}, \vec{u})$$

Intuitively, with the above rules we traverse all types checking the conditions (C0), (C1′), (C2′) described in ($\mathbf{M}_\exists$). If we manage to reach the last type, and if $A \in \vec{x}$, then we know the condition is satisfied and mark the type:

$$\mathsf{Marked}(\vec{x}) \leftarrow \mathsf{MarkedUntil}_\alpha(\vec{x}, \vec{z}), A \in \vec{x}, \mathsf{last}(\vec{z})$$

(IX) 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:

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

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:

$$\begin{aligned}
\mathsf{Proj}^{i-1}(x, \vec{y}) &\leftarrow \mathsf{Proj}^i(x, \vec{y}, 1), B_i(x) && \text{if } B_i \in \mathsf{N_C} \\
\mathsf{Proj}^{i-1}(a, \vec{y}) &\leftarrow \mathsf{Proj}^i(a, \vec{y}, 1) && \text{if } B_i = \{a\} \\
\mathsf{Proj}^{i-1}(x, \vec{y}) &\leftarrow \mathsf{Proj}^i(x, \vec{y}, 0), x \neq a && \text{if } B_i = \{a\} \\
\mathsf{Proj}^{i-1}(x, \vec{y}) &\leftarrow \mathsf{Proj}^i(x, \vec{y}, 0), \overline{B}_i(x) && \text{if } B_i \notin \Sigma \\
\mathsf{Proj}^{i-1}(x, \vec{y}) &\leftarrow \mathsf{Proj}^i(x, \vec{y}, 0), \mathit{not}\ B_i(x) && \text{if } B_i \in \Sigma
\end{aligned}$$

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 $\mathsf{Proj}^0(a)$ represents 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 rewriting of the instance query $(\mathcal{T}, \Sigma, q)$ into the program P. In case $\Sigma = \emptyset$, our rewriting does not use not in rule bodies. If nominals are not present in $\mathcal{T}$, we also don't use the predicate $\neq$. This construction together with Theorems 1 and 2 yields our main result:

Theorem 3. *For an instance query $(\mathcal{T}, \Sigma, q)$, where $\mathcal{T}$ is an $\mathcal{ALCHIO}$ TBox, we can build in polynomial time a query (P, q), where P may use disjunction and negation, such that:*

(i) The certain answers to $(\mathcal{T}, \Sigma, q)$ and (P, q) coincide for any given ABox $\mathcal{A}$ over the signature of $\mathcal{T}$.

(ii) If $\Sigma = \emptyset$, then P is a positive program.

(iii) If $\Sigma = \emptyset$ and $\mathcal{T}$ is an $\mathcal{ALCHI}$ TBox, then P is a positive program with no occurrences of the $\neq$ predicate.

The above encoding employs disjunction, and possibly negation as failure. Entailment of ground atoms already in positive disjunctive programs is coNEXPTIME-complete [Eiter *et al.*, 1997], which does not match the EXPTIME-completeness of satisfiability of $\mathcal{ALCHIO}$ KBs with (or without) closed predicates. However, we employ disjunction and negation 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 ("stratified") into programs P_1, P_2, P_3 as follows:

- P_1 consists of all rules in (I) and (II), except the constraints. P_1 is a positive disjunctive program with at most two variables in each rule.
- P_2 consists of the constraints in (II), and the rules in (V). Intuitively, the disjunction-free P_2 ensures that P_1 generates a proper core interpretation, and computes the types realized in it.
- P_3 consists of the remaining rules, and is disjunction-free.

Note that P_2 and P_3 do 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 $P_2 \cup P_3$. The program P_2 only depends on P_1, i.e. none

of relation symbols in P_2 occurs in the head of a rule in P_3. The negative atoms of P_2 only involve relations that are fully defined by P_1, i.e. such relation do not appear in the head of rule in P_2. Similarly, the negative atoms of P_3 only involve relations that are fully defined by $P_1 \cup P_2$. 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 stable models of $P \cup F$:

(S1) Compute a minimal model I_1 of $P_1 \cup F$.

(S2) Compute the least model I_2 of $I_1 \cup P_2^{I_1}$. If I_2 does not exist due to a constraint violation, then return *failure*.

(S3) Compute the least model I_3 of $I_2 \cup P_2^{I_2}$. Again, if I_3 does not exist, then return *failure*. Otherwise, output I_3.

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 steps (S2) and (S3) is feasible in (deterministic) exponential time, because $P_2^{I_1}$ and the subsequent $P_2^{I_2}$ are ground disjunction-free positive programs of exponential size. 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.

We remark that if there are no closed predicates, i.e. in case $\Sigma = \emptyset$, the resulting program has an even simpler structure. It contains all rules of P that do not have not in rule bodies, and it can be partitioned into *two* programs:

- a disjunctive program P_1 with at most two variables in each rule, which can include all positive rules in (I) and (II);
- a disjunction-free P_2 that does not define any relations in P_1, which consists all the remaining rules.

Analogously as above, we can evaluate P over a given set F of facts, by traversing the minimal models I_1 of $P_1 \cup F$, and for each I_1, verifying in (deterministic) exponential time if it can be extended to a model of P_2. More details about this case, but without nominals, i.e. for the DL $\mathcal{ALCHI}$, can be found in [Ahmetaj *et al.*, 2016].

5 Discussion

We note that $\neq$-free positive programs are not expressive enough to capture instance queries $(\mathcal{T}, \emptyset, q)$ when $\mathcal{T}$ has nominals. This follows from the following observation. For any positive $\neq$-free program P and a set of facts F, if $P \cup F$ has a model, then also $P \cup F'$ has a model, were F' is obtained from F by renaming its constants with fresh ones that don't occur in $P \cup F$. However, this property cannot be recast to $\mathcal{ALCHIO}$. Take the TBox $\mathcal{T} = \{A \sqsubseteq \{a\}\}$ and observe that $\mathcal{T}$ is consistent w.r.t. the ABox $\mathcal{A}_1 = \{A(a)\}$, but is inconsistent w.r.t. the ABox $\mathcal{A}_2 = \{A(b)\}$.

We have presented our results for $\mathcal{ALCHIO}$, but they also apply to $\mathcal{SHIO}$, using standard techniques to eliminate transitivity axioms (see, e.g., [Hustadt *et al.*, 2007]). Moreover, the results can be easily generalized, e.g., to DL-safe rules of [Motik *et al.*, 2005], or the *quantifier-free* CQs like in [Lutz *et al.*, 2015]. These queries are syntactically restricted to ensure that the relevant variable assignments only map into individuals of the input ABox. Under common assumptions in complexity theory, our translation cannot be generalized to CQs, while remaining polynomial. This is because cautious inference from a disjunctive program with negation is in coNEXPTIME$^{\text{NP}}$, but entailment of (Boolean) CQs is 2EXPTIME-hard already for the DL $\mathcal{ALCI}$ [Lutz, 2007].

Acknowledgments

In memory of Helmut Veith (1971-2016).

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

References

[Ahmetaj *et al.*, 2016] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Šimkus. Polynomial disjunctive datalog rewritings of instance queries in expressive description logics. In *Proc. of DL 2016*, 2016.

[Baader *et al.*, 2007] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, second edition, 2007.

[Bienvenu and Ortiz, 2015] Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Reasoning Web*, volume 9203 of *Lecture Notes in Computer Science*, pages 218–307. Springer, 2015.

[Bienvenu *et al.*, 2014] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.

[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. Autom. Reasoning*, 39(3):385–429, 2007.

[Dantsin *et al.*, 2001] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[Eiter *et al.*, 1997] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

[Eiter *et al.*, 2012a] Thomas Eiter, Magdalena Ortiz, and Mantas Šimkus. Conjunctive query answering in the description logic SH using knots. *J. Comput. Syst. Sci.*, 78(1):47–85, 2012.

[Eiter *et al.*, 2012b] Thomas Eiter, Magdalena Ortiz, Mantas Šimkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for Horn-SHIQ plus rules. In *Proc. of AAAI 2012*. AAAI Press, 2012.

[Franconi *et al.*, 2011] Enrico Franconi, Yazmin Angélica Ibáñez-García, and Inanç Seylan. Query answering with DBoxes is hard. *Electr. Notes Theor. Comput. Sci.*, 278:71–84, 2011.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP/SLP 1988*. MIT Press, 1988.

[Gottlob and Schwentick, 2012] Georg Gottlob and Thomas Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In *Proc. of KR 2012*. AAAI Press, 2012.

[Gottlob *et al.*, 2014a] Georg Gottlob, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, Thomas Schwentick, and Michael Zakharyaschev. The price of query rewriting in ontology-based data access. *Artif. Intell.*, 213:42–59, 2014.

[Gottlob *et al.*, 2014b] Georg Gottlob, Marco Manna, and Andreas Pieris. Polynomial combined rewritings for existential rules. In *Proc. of KR 2014*. AAAI Press, 2014.

[Gottlob *et al.*, 2015] Georg Gottlob, Marco Manna, and Andreas Pieris. Polynomial rewritings for linear existential rules. In *Proc. of IJCAI 2015*. AAAI Press, 2015.

[Hustadt *et al.*, 2007] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in description logics by a reduction to disjunctive datalog. *J. Autom. Reasoning*, 39(3):351–384, 2007.

[Kaminski *et al.*, 2014a] Mark Kaminski, Yavor Nenov, and Bernardo Cuenca Grau. Computing datalog rewritings for disjunctive datalog programs and description logic ontologies. In *Proc. of RR 2014*, pages 76–91, 2014.

[Kaminski *et al.*, 2014b] Mark Kaminski, Yavor Nenov, and Bernardo Cuenca Grau. Datalog rewritability of disjunctive datalog programs and its applications to ontology reasoning. In *Proc. of AAAI 2014*, pages 1077–1083, 2014.

[Kontchakov *et al.*, 2011] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to ontology-based data access. In *Proc. of IJCAI 2011*. IJCAI/AAAI, 2011.

[Lutz *et al.*, 2013] Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-based data access with closed predicates is inherently intractable(sometimes). In *Proc. of IJCAI 2013*. IJCAI/AAAI, 2013.

[Lutz *et al.*, 2015] Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-mediated queries with closed predicates. In *Proc. of IJCAI 2015*. IJCAI/AAAI, 2015.

[Lutz, 2007] Carsten Lutz. Inverse roles make conjunctive queries hard. In *Proc. of DL 2007*. CEUR-WS.org, 2007.

[Motik *et al.*, 2005] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *J. Web Sem.*, 3(1):41–60, 2005.

[Ngo *et al.*, 2015] Nhung Ngo, Magdalena Ortiz, and Mantas Šimkus. The combined complexity of reasoning with closed predicates in description logics. In *Proc. of DL 2015*. CEUR-WS.org, 2015.

[Ortiz *et al.*, 2010] Magdalena Ortiz, Sebastian Rudolph, and Mantas Šimkus. Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In *Proc. of KR 2010*. AAAI Press, 2010.

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

[Seylan *et al.*, 2009] Inanç Seylan, Enrico Franconi, and Jos de Bruijn. Effective query rewriting with ontologies over DBoxes. In *Proc. of IJCAI 2009*, 2009.

[Trivela *et al.*, 2015] Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos B. Stamou. Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Sem.*, 33:30–49, 2015.