Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)
Main Track

Datalog Rewritability and Data Complexity of $\mathcal{ALCHOIF}$ with Closed Predicates

Tomasz Gogacz[1] , **Sanja Lukumbuzya**[2] , **Magdalena Ortiz**[2] , **Mantas Šimkus**[2]
[1]Institute of Informatics, University of Warsaw, Poland
[2]Institute of Logic and Computation, TU Wien, Austria
t.gogacz@mimuw.edu.pl, {lukumbuzya, ortiz}@kr.tuwien.ac.at, simkus@dbai.tuwien.ac.at

Abstract

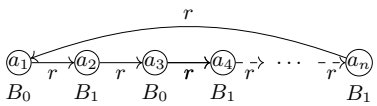
We study the relative expressiveness of ontology-mediated queries (OMQs) formulated in the expressive Description Logic $\mathcal{ALCHOIF}$ extended with closed predicates. In particular, we present a polynomial-time translation from OMQs into Datalog with negation under the stable model semantics, the formalism that underlies Answer Set Programming. This is a novel and non-trivial result: the considered OMQs are not only non-monotonic but also feature a tricky combination of nominals, inverse roles, and role functionality. We start with atomic queries and then lift our approach to a large class of first-order queries where quantification is "guarded" by closed predicates. Our translation is based on a characterization of the query answering problem via integer programming, and a specially crafted program in Datalog with negation that finds solutions to dynamically generated systems of integer inequalities. As an important by-product of our translation, we get that the query answering problem is co-NP-complete in data complexity for the considered class of OMQs. Thus, answering these OMQs in the presence of closed predicates is not harder than answering them in the standard setting. This is not obvious as closed predicates are known to increase data complexity for some existing ontology languages.

1 Introduction

Description Logics (DLs) are a prominent family of knowledge representation languages suitable for capturing and reasoning about information with a complex structure. For instance, DLs play a key role in *Ontology-Based Data Access (OBDA)*, a data integration approach where a DL-based *ontology* in combination with a reasoner acts as a mediator between the users and the possibly heterogeneous data sources. A key reasoning task in OBDA is answering *ontology-mediated queries (OMQs)*. An OMQ is a pair $(\mathcal{T}, Q)$ that couples an ontology $\mathcal{T}$ (or a *TBox*) expressed in some DL and a database query Q. Given a set $\mathcal{A}$ of facts (or an *ABox*), answering $(\mathcal{T}, Q)$ requires computing the intersection of answers to Q over all structures (essentially, databases) that contain $\mathcal{A}$ and satisfy the ontological axioms in $\mathcal{T}$.

Most DLs are fragments of first-order logic, which leads to the *open-world assumption* in OBDA and other applications of DLs to data management. Intuitively, the open-world assumption states that everything that is not forbidden is possible. This sometimes rules out intuitive common-sense inferences, and thus has spurred research efforts to combine the open-world assumption with the *closed-world assumption* that is made in the classical database setting. One of the basic tools to achieve this in DLs are the so-called *closed predicates* (Franconi, Ibáñez-García, and Seylan 2011; Lutz, Seylan, and Wolter 2013), which allow us to specify which ontological predicates have extensions that are determined solely by the data.

In this paper, we focus on an extension of the expressive DL $\mathcal{ALCHOIF}$ with closed predicates. $\mathcal{ALCHOIF}$ is a close relative of the W3C standard OWL 2 ontology language, and it extends the basic DL $\mathcal{ALC}$ with role hierarchies, nominals, inverse roles, and role functionality. The combination of the latter three constructors is known to be particularly tricky, and their interaction causes NEXPTIME-hardness of some of the basic reasoning problems like ontology satisfiability (Tobies 2000). For comparison, this problem is in EXPTIME if one of the three is omitted. The extension of $\mathcal{ALCHOIF}$ with closed predicates is a very powerful language that allows us to easily express many natural queries that cannot be expressed in traditional query languages like first-order logic or Datalog. One such example is determining, given a database and a unary relation A, whether the number of objects in A is odd. This can be expressed through a Boolean OMQ $(\mathcal{T}, \bot)$, where $\mathcal{T} = \{A \equiv B_1 \sqcup B_2, B_1 \sqcap B_2 \sqsubseteq \bot, B_1 \sqsubseteq \exists r.B_2, B_2 \sqsubseteq \exists r.B_1, (\text{func } r), (\text{func } r^-)\}$ and A is closed. Let $a_1, \ldots, a_n$ be the elements in A. The axioms in $\mathcal{T}$ force models to partition $a_1, \ldots, a_n$ into two sets B_1 and B_2. Further, every element in B_1 has an r-arc to an element in B_2 and vice versa. Due to the functionality of r and r^-, each element has exactly one ingoing and one outgoing r-arc. As A is a closed predicate, this forces the existence of a cycle in models of $\mathcal{T}$ and $\mathcal{A}$ of the following form:



Such a cycle exists if and only if n is even. Which means that for odd n, the query is entailed. Other queries that can be expressed using $\mathcal{ALCHOIF}$ ontologies with closed predicates include comparing cardinalities of two database

relations, e.g., checking whether there are twice as many elements in relation A than in relation B.

The main motivation behind this paper is to understand the *relative expressiveness* of OMQs based on this DL, especially compared to more standard database languages like various extensions of Datalog. Our main result is a polynomial-time translation that allows us to convert an $\mathcal{ALCHOIF}$ TBox with closed predicates into a Datalog program with negation under the stable model semantics. When data is considered, this program can be used to answer any instance query. In particular, given a TBox $\mathcal{T}$ and a set Σ of predicates that are considered closed, we obtain in polynomial time a program $P^{\mathcal{T},\Sigma}$ such that the knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ and the program $P^{\mathcal{T},\Sigma} \cup \mathcal{A}$ have the same atomic consequences. We thus establish that instance queries mediated by $\mathcal{ALCHOIF}$ ontologies can be rewritten into Datalog with stable negation. This result extends to a much larger class of *safe-range OMQs*, which support first-order queries where quantification is "guarded" by closed predicates.

An important consequence of our results is that the considered class of OMQs is co-NP-complete in data complexity, which follows from the complexity of Datalog with negation under the stable model semantics. This is a positive result, as it shows that adding closed predicates to $\mathcal{ALCHOIF}$ does not increase the data complexity. We note that this is not obvious, as closed predicates are known to increase the data complexity in some cases, e.g., for ontology languages based on existential rules (Benedikt et al. 2016).

Rewritability of DLs into traditional database languages has received significant attention. For lightweight DLs, rewritability into FO queries has been studied extensively and it underlies actual OBDA systems (Calvanese et al. 2007). For expressive DLs, the target language of the rewritings is usually a variant of Datalog (Bienvenu et al. 2014; Gottlob and Schwentick 2012). For example, an exponential rewriting for $\mathcal{SHIQ}$ into disjunctive Datalog was presented in (Hustadt, Motik, and Sattler 2007), a polynomial-time Datalog encoding of Horn-$\mathcal{SHIQ}$ was proposed in (Ortiz, Rudolph, and Šimkus 2010) and (Ahmetaj, Ortiz, and Šimkus 2020) propose a polynomial-time translation for $\mathcal{ALCHOI}$ with closed predicates. Our translation uses a very different approach compared to other translations in the literature. Inspired by works on finite satisfiability (Calvanese 1996; Lutz, Sattler, and Tendera 2005; Pratt-Hartmann 2005), we characterize models of $\mathcal{ALCHOIF}$ KBs with closed predicates by means of integer programming. We then present a Datalog program that performs ontology-mediated query answering by computing solutions to a dynamically generated system of inequalities.

The rest of the paper is structured as follows. Section 2 reviews some basic notions in DLs. In Section 3, we present the integer programming characterization of $\mathcal{ALCHOIF}$ satisfiability with closed predicates. Based on this, Section 4 shows how to construct a Datalog program with stable negation for deciding this satisfiability problem. Finally, in Section 5 we explain how this program can be augmented for answering certain types of OMQs and we present our complexity results.

An extended version of this paper can be found at https://dbai.tuwien.ac.at/staff/simkus/papers/kr20-data.pdf

2 Preliminaries

We assume countably infinite sets $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_I}$ of unary predicate symbols (*concept names*), binary predicate symbols (*role names*), and *constants*, respectively. *Roles* are expressions of the form p and p^-, where $p \in \mathsf{N_R}$. We let $\mathsf{N_R^+}$ be the set of all roles. With a slight abuse of notation, we write r^- to denote p^- if $r = p$, and p if $r = p^-$, for $p \in \mathsf{N_R}$. The role r^- is the *inverse* of r. Given a set R of roles, R^- denote the set $\{r^- : r \in R\}$. $\mathcal{ALCHOIF}$ *concepts* are defined according to the following syntax: $C := \top \mid \bot \mid A \mid \{a\} \mid \neg C \mid C \sqcup C \mid C \sqcap C \mid \forall r.C \mid \exists r.C$, where $A \in \mathsf{N_C}$, r is a role, and $a \in \mathsf{N_I}$. *Nominals* are concepts of the form $\{a\}$, where $a \in \mathsf{N_I}$. An *ABox* is a finite set of *assertions* of the form $A(a)$, $\neg A(a)$, $p(a, b)$ or $\neg p(a, b)$, where $a, b \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, and $p \in \mathsf{N_R}$. An expression $C \sqsubseteq D$, where C and D are concepts, is a *concept inclusion*, and an expression $r \sqsubseteq s$, where r and s are roles, is a *role inclusion*. A *functionality assertion* is an expression (func r), where r is a role. A *(TBox) axiom* is a concept inclusion, a role inclusion, or a functionality assertion, and a *TBox* is a finite set of axioms. A *knowledge base (with closed predicates)* (KB) is a triple $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, where $\mathcal{T}$ is a TBox, $\mathcal{A}$ is an ABox, and $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ is a set of *closed predicates*. For a TBox, an ABox, or a KB $\mathcal{X}$, we denote by $\mathsf{N_C}(\mathcal{X})$, $\mathsf{N_R}(\mathcal{X})$ the set of concept and role names occurring in $\mathcal{X}$, and by $\mathsf{N_R^+}(\mathcal{X})$ the set of roles occurring in $\mathcal{X}$ and their inverses.

An interpretation is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set called the *domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function* that assigns to each $a \in \mathsf{N_I}$ a domain element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, each $A \in \mathsf{N_C}$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and each $r \in \mathsf{N_R}$ a set $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The extension of the interpretation function to the remaining concepts and roles is defined in the standard way (Baader et al. 2003). Note that, as every predicate has an arity of at most two, interpretations can be seen as a labeled directed graph, where each vertex represents a domain element d and is labeled with the concept names and nominals d participates in, and each edge between two vertices d_1 and d_2 is labeled by roles and represents a pair (d_1, d_2) in their extensions. A concept or a role inclusion $Q_1 \sqsubseteq Q_2$ is satisfied by an interpretation $\mathcal{I}$ if $Q_1^{\mathcal{I}} \subseteq Q_2^{\mathcal{I}}$. A functionality assertion (func r) is satisfied by $\mathcal{I}$ if for all $c \in \Delta^{\mathcal{I}}$, $(c, d_1) \in r^{\mathcal{I}}$ and $(c, d_2) \in r^{\mathcal{I}}$ implies $d_1 = d_2$ and $\mathcal{I}$ satisfies an assertion $Q(\vec{a})$ if $\vec{a} \in Q^{\mathcal{I}}$. We note that we make the *Standard Name Assumption (SNA)*, which is common when dealing with closed predicates and which forces us to interpret every constant as itself, i.e., $a^{\mathcal{I}} = a$, for all $\mathcal{I}$ and all constants a. We say that $\mathcal{I}$ satisfies a TBox $\mathcal{T}$ (resp. ABox $\mathcal{A}$) if it satisfies all axioms in $\mathcal{T}$ (resp. assertions in $\mathcal{A}$). We say that $\mathcal{I}$ satisfies an ABox $\mathcal{A}$ under the closed predicates Σ, in symbols $\mathcal{I} \models_\Sigma \mathcal{A}$, if $\mathcal{I} \models \mathcal{A}$ and $\vec{a} \in Q^{\mathcal{I}}$ implies $Q(\vec{a}) \in \mathcal{A}$, for all $Q \in \Sigma$. We say that $\mathcal{I}$ satisfies a KB $\mathcal{K}$, in symbols $\mathcal{I} \models \mathcal{K}$, if $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models_\Sigma \mathcal{A}$.

3 KB Satisfiability via Integer Programming

In this section, we devise a procedure that decides the satisfiability of $\mathcal{ALCHOIF}$ KBs with closed predicates using integer programming techniques. Our approach is closely related to techniques in (Calvanese 1996; Lutz, Sattler, and

Tendera 2005; Pratt-Hartmann 2005) that reduce the *finite satisfiability problem* to finding integer solutions to a system of linear inequalities. We reuse large parts of the inequality systems in (Gogacz et al. 2020), which (among other results) shows that deciding whether a given $\mathcal{ALCHOIF}$ TBox has a model in which *some* input predicates have finite extensions can be characterized via integer programming. For this paper we need to enable ABoxes and closed predicates: we next show that we can correctly characterize the problem of satisfiability of $\mathcal{ALCHOIF}$ KBs with closed predicates as a system of linear inequalities together with some side conditions. This characterization is the basis of our translation from queries mediated by $\mathcal{ALCHOIF}$ TBoxes into Datalog with negation under the stable model semantics.

We begin by introducing a couple of key notions. Let $\mathsf{N}_\mathsf{C}^+ = \mathsf{N}_\mathsf{C} \cup \{\top, \bot\} \cup \{\{c\} : c \in \mathsf{N}_\mathsf{I}\}$ be the set of *basic concepts* and let $\mathsf{N}_\mathsf{C}^+(\mathcal{K})$ denote the concepts in N_C^+ that occur in a KB $\mathcal{K}$. A *tile* τ for $\mathcal{K}$ is a description of a domain element together with its relevant neighborhood. Intuitively, tiles tell us which basic concepts domain elements participate in as well as the kind of neighbors they have in a model of $\mathcal{K}$. A *mosaic* for $\mathcal{K}$ is a function N that assigns to each tile a multiplicity. Mosaics satisfy certain conditions that ensure that it is possible to build a model of $\mathcal{K}$ by taking, for each tile τ, $N(\tau)$ domain elements that fit the description given by τ. Deciding satisfiability of $\mathcal{K}$ thus amounts to deciding the existence of a mosaic for $\mathcal{K}$.

To simplify our presentation, we make a couple of assumptions (w.l.o.g) regarding the kind of knowledge bases we consider. First, we assume that every concept/role name occurring in the knowledge base also occurs in its TBox. We also assume that TBoxes are given in *normal form* in which all inclusions are of the following types:

$$B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m,$$
$$B_1 \sqsubseteq \exists r.B_2, \qquad B_1 \sqsubseteq \forall r.B_2, \qquad r \sqsubseteq s, \qquad (\text{func } r),$$

where $\{B_1, \ldots, B_m\} \subseteq \mathsf{N}_\mathsf{C}^+$, $k > 1$, $m \geq k$, and $\{r, s\} \subseteq \mathsf{N}_\mathsf{R}^+$. Moreover, TBoxes are assumed to be closed under role inclusions as follows:

- $p \sqsubseteq p \in \mathcal{T},$ for each $p \in \mathsf{N}_\mathsf{R}$ occurring in $\mathcal{T}$
- if $r \sqsubseteq s \in \mathcal{T},$ then $r^- \sqsubseteq s^- \in \mathcal{T}$
- if $r_1 \sqsubseteq r_2 \in \mathcal{T}$ and $r_2 \sqsubseteq r_3 \in \mathcal{T},$ then $r_1 \sqsubseteq r_3 \in \mathcal{T}$
- if $r \sqsubseteq s \in \mathcal{T}$ and $(\text{func } s) \in \mathcal{T}$, then $(\text{func } r) \in \mathcal{T}$.

We next formally define the notions of tiles and mosaics.

Definition 1. Given an $\mathcal{ALCHOIF}$ knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, a *type* for $\mathcal{K}$ is any set T such that

1. $T \subseteq \mathsf{N}_\mathsf{C}^+(\mathcal{K})$;
2. $\top \in T$ and $\bot \notin T$;
3. for all $a, b \in \mathsf{N}_\mathsf{I}(\mathcal{K})$, if $\{a\}, \{b\} \in T$, then $a = b$.

We denote the set of all types for $\mathcal{K}$ by Types($\mathcal{K}$). A type lists the basic concepts that a domain element participates in. The intuition behind the first two conditions is obvious and the third condition arises due to the SNA.

Definition 2. Given an $\mathcal{ALCHOIF}$ KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, a *tile* for $\mathcal{K}$ is a tuple $\tau = (T, \rho)$, where $T \in \text{Types}(\mathcal{K})$ and ρ is a set of pairs (R, T'), where $R \subseteq \mathsf{N}_\mathsf{R}^+(\mathcal{K})$, $T' \in \text{Types}(\mathcal{K})$ and the following conditions are satisfied:

1. $|\rho| \leq |\mathcal{T}|$
2. If $B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m \in \mathcal{T}$ and $\{B_1, \ldots, B_{k-1}\} \subseteq T$, then $\{B_k, \ldots, B_m\} \cap T \neq \emptyset$
3. If $A \sqsubseteq \exists r.B \in \mathcal{T}$ and $A \in T$, then there is $(R, T') \in \rho$ such that $r \in R$ and $B \in T'$
4. For all $(R, T') \in \rho$, the following hold:
   (a) If $A \sqsubseteq \forall r.B \in \mathcal{T}$, $A \in T$ and $r \in R$, then $B \in T'$
   (b) If $A \sqsubseteq \forall r.B \in \mathcal{T}$, $A \in T'$ and $r^- \in R$, then $B \in T$
   (c) If $r \sqsubseteq s \in \mathcal{T}$ and $r \in R$, then $s \in R$
5. If $(\text{func } r) \in \mathcal{T}$, then $|\{(R, T') \in \rho : r \in R\}| \leq 1$
6. If $A(b) \in \mathcal{A}$ and $\{b\} \in T$, then $A \in T$
7. If $\neg A(b) \in \mathcal{A}$ and $\{b\} \in T$, then $A \notin T$
8. For all $(R, T') \in \rho$, the following hold:
   (a) If $p(a, b) \in \mathcal{A}$, $\{p \sqsubseteq r, (\text{func } r)\} \subseteq \mathcal{T}$, $\{a\} \in T$ and $r \in R$, then $\{b\} \in T'$
   (b) If $p(a, b) \in \mathcal{A}$, $\{p \sqsubseteq r, (\text{func } r^-)\} \subseteq \mathcal{T}$, $\{b\} \in T$, and $r^- \in R$, then $\{a\} \in T'$
   (c) If $\neg p(a, b) \in \mathcal{A}$, $r \sqsubseteq p \in \mathcal{T}$, $\{a\} \in T$, and $r \in R$, then $\{b\} \notin T'$
   (d) If $\neg p(a, b) \in \mathcal{A}$, $r \sqsubseteq p^- \in \mathcal{T}$, $\{b\} \in T$, and $r \in R$, then $\{a\} \notin T'$
9. If $A \in \Sigma \cap \mathsf{N}_\mathsf{C}$ and $A \in T$, then there exists $c \in \mathsf{N}_\mathsf{I}$ such that $\{c\} \in T$ and $A(c) \in \mathcal{A}$
10. If $r \in \Sigma \cap \mathsf{N}_\mathsf{R}$, then for all $(R, T') \in \rho$ with $r \in R$, there exist $c, d \in \mathsf{N}_\mathsf{I}$ such that $\{c\} \in T$, $\{d\} \in T'$ and $r(c, d) \in \mathcal{A}$.

A tile (T, ρ) describes a domain element d that participates in the basic concepts in T and that, for each $(R, T') \in \rho$, has an arc labeled by R to a domain element that participates in the basic concepts given by T'. It is crucial to note that, in general, ρ does not describe all the connections that d may have in a model of $\mathcal{K}$, i.e., in addition to those described by ρ, d can have further connections. This kind of encoding allows us to keep the number of different tiles polynomial in the size of the Abox $\mathcal{A}$, and their size independent of $\mathcal{A}$, which plays an important role in defining a polynomial and data-independent Datalog translation. We briefly explain the intuitions behind the conditions. Conditions 1-5 are inherited from (Gogacz et al. 2020) and ensure that the description of d is consistent with the statements in $\mathcal{T}$, from the perspective of d. Condition 2 ensures the satisfaction of axioms of the type $B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m \in \mathcal{T}$, condition 3 guarantees that d has a witness for every axiom $\exists R.B \in \mathcal{T}$, conditions 4 (a)-(c) ensure that d and its neighbors respect axioms of the type $A \sqsubseteq \forall r.B$ and $r \sqsubseteq s$, and condition 5 ensures that the functionality assertions in $\mathcal{T}$ are not violated. We further add the conditions that ensure that the description of d is compatible with the assertions in $\mathcal{A}$ (conditions 6-8), and those that ensure that the closed predicates are respected (conditions 9-10).

We next define mosaics for which are functions that tell us, for each tile τ, how many instances of τ we need to build a model. Since some tiles might need to be instantiated infinitely many times we introduce a new value $\aleph_0$ that is greater than any natural number and we extend the operations $\cdot$ and $+$ as follows: $\aleph_0 \cdot \aleph_0 = \aleph_0 + \aleph_0 = \aleph_0 + 0 = 0 + \aleph_0 = \aleph_0 + n = n + \aleph_0 = \aleph_0 \cdot n = n \cdot \aleph_0 = \aleph_0$, for all $n \in \mathbb{N} \setminus \{0\}$, and $0 \cdot \aleph_0 = \aleph_0 \cdot 0 = 0$. We let $\mathbb{N}^* = \mathbb{N} \cup \{\aleph_0\}$.

Definition 3. Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIF}$ KB. A *mosaic* for $\mathcal{K}$ is a function $N : \text{Tiles}(\mathcal{K}) \to \mathbb{N}^*$ such that:

M1. For every $\{c\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$: $\sum_{\substack{(T,\rho)\in\text{Tiles}(\mathcal{K}),\\ \{c\}\in T}} N((T,\rho)) = 1$

M2. The following inequality is satisfied: $\sum_{\tau\in\text{Tiles}(\mathcal{K})} N(\tau) \geq 1$

M3. For every pair $T, T' \in \text{Types}(\mathcal{K})$ and every $R \subseteq \mathsf{N}_\mathsf{R}^+(\mathcal{K})$ with $r \in R$ and $(\text{func } r^-) \in \mathcal{T}$, the following holds:

$$\sum_{\substack{(T,\rho)\in\text{Tiles}(\mathcal{K}),\\ (R,T')\in\rho}} N((T,\rho)) \leq \sum_{\substack{(T',\rho')\in\text{Tiles}(\mathcal{K}),\\ (R^-,T)\in\rho'}} N((T',\rho'))$$

M4. For all $(T,\rho) \in \text{Tiles}(\mathcal{K})$ and $(R,T') \in \rho$ the following holds: if $N((T,\rho)) > 0$, then there exists ρ' such that $(T',\rho') \in \text{Tiles}(K)$ and $N((T',\rho')) > 0$.

M5. For all $\{a\}, \{b\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$ and all $A, B \in \mathsf{N}_\mathsf{C}(\mathcal{K})$, if there exist $p, r \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$ for which any of the following conditions hold:

(a) $p(a,b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$ and $A \sqsubseteq \forall r.B \in \mathcal{T}$,

(b) $p(b,a) \in \mathcal{A}$, $p \sqsubseteq r^- \in \mathcal{T}$ and $A \sqsubseteq \forall r.B \in \mathcal{T}$,

(c) $p(a,b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$ and $A \sqsubseteq \exists r.B \in \mathcal{T}$ and $(\text{func } r) \in \mathcal{T}$, or

(d) $p(b,a) \in \mathcal{A}$, $p \sqsubseteq r^- \in \mathcal{T}$ and $A \sqsubseteq \exists r.B \in \mathcal{T}$ and $(\text{func } r) \in \mathcal{T}$,

we have that the following implication holds:

$$\sum_{\substack{(T,\rho)\in\text{Tiles}(\mathcal{K}),\\ \{a\}\in T, A\in T}} N((T,\rho)) > 0 \text{ implies } \sum_{\substack{(T',\rho')\in\text{Tiles}(\mathcal{K}),\\ \{b\}\in T', B\in T'}} N((T',\rho')) > 0$$

We can see a mosaic N for $\mathcal{K}$ as a recipe for building a model of $\mathcal{K}$ by instantiating tiles according to the multiplicities given by N. The first condition tells us that in a model of $\mathcal{K}$, for all constants c occurring in $\mathcal{K}$ there can be exactly one domain element that participates $\{c\}$ – namely c itself. The second condition makes sure that at least one tile is instantiated as we do not allow interpretations with empty domains. The condition M3 ensures that we have enough domain elements to satisfy the functionality assertions in $\mathcal{T}$. More precisely, assume we are given types T, T' and $R \subseteq \mathsf{N}_\mathsf{R}^+(\mathcal{K})$, where a role r whose inverse is functional occurs in R. Let n be the number of domain elements in a model of $\mathcal{K}$ that participate in basic concepts in T and have an outgoing arc labeled by R to some neighbor of type T'. Since r^- is functional, each domain element can "accept" at most one incoming arc labeled by R, or equivalently, has at most one outgoing arc labeled by R^-. Thus, to build a model of $\mathcal{K}$, there must be n or more elements of type T' that have an outgoing arc to an element of type T that is labeled by R^-. The condition M4 says that if we obtain a domain element d by instantiating a tile (T,ρ) and ρ asserts the existence of some neighbors of d, we can also instantiate tiles to provide suitable neighbors for d. Condition M5 relates to the following situation. Assume we have $p(a,b) \in \mathcal{A}$ asserting that, in a model of $\mathcal{K}$, a has as a p-neighbor the constant b and assume a participates in a concept name A. Now assume there are axioms in $\mathcal{T}$ that allows us to infer that all p-neighbors of a must participate in a concept name B. We can conclude that b must participate in B. Constant a can "send" such a message to b either via universal axioms in $\mathcal{T}$ or via existential axioms and functionality assertions in $\mathcal{T}$. Conditions M5 (a)-(d) represent different ways a can communicate information to b.

The following result establishes the connection between the existence of mosaics and the satisfiability of $\mathcal{ALCHOIF}$ KBs with closed predicates.

Theorem 1. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIF}$ KB. $\mathcal{K}$ is satisfiable if and only if there exists a mosaic for $\mathcal{K}$ and $\mathcal{A}$ satisfies the following conditions:*

1. *for all $r \in \Sigma \cap \mathsf{N}_\mathsf{R}$ and $s \sqsubseteq r \in \mathcal{T}$, if $s(a,b) \in \mathcal{A}$, then $r(a,b) \in \mathcal{A}$, and*
2. *for all $r \in \mathsf{N}_\mathsf{R}^+(\mathcal{A})$ with $(\text{func } r) \in \mathcal{T}$ and all $a \in \mathsf{N}_\mathsf{I}(\mathcal{A})$, the set $\{b : p(a,b) \in \mathcal{A}, p \sqsubseteq r \in \mathcal{T}\} \cup \{b : p(b,a) \in \mathcal{A}, p^- \sqsubseteq r \in \mathcal{T}\}$ has at most 1 element.*

Intuitively, the two conditions in Theorem 1 ensure that $\mathcal{A}$ itself respects the closed predicates and functionality assertions. The first condition states that for each role $r \in \Sigma$, if $p(a,b) \in \mathcal{A}$ and $p \sqsubseteq r \in \mathcal{T}$, then also $r(a,b) \in \mathcal{A}$. If this is not the case, every interpretation satisfying $p \sqsubseteq r$ violates the closed predicates and as such $\mathcal{K}$ does not have a model. The second condition checks for each constant a in $\mathcal{A}$ and each functional role r, whether $\mathcal{A}$ asserts the existence of more than one r-neighbor of a. If that is the case, then $\mathcal{K}$ also does not have a model. Thus, deciding whether $\mathcal{K}$ is satisfiable amounts to checking whether $\mathcal{A}$ satisfies the given conditions and whether there exists a mosaic for $\mathcal{K}$. To decide the latter, we build a system of linear inequalities with implications whose solutions over $\mathbb{N}^*$ define mosaics for $\mathcal{K}$.

Formally, a *system of linear inequalities* is a pair $(V, \mathcal{E})$, where V is a set of variables and $\mathcal{E}$ is a set of inequalities of the form $a_1 \cdot x_1 + \cdots + a_n \cdot x_n + c \leq b_1 \cdot y_1 + \cdots + b_m \cdot y_m$, where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are positive integers, c is a possibly negative integer, and $x_1, \ldots, x_n, y_1, \ldots, y_m \in V$. For our purposes, we consider *enriched systems of linear inequalities* that are tuples $(V, \mathcal{E}, I)$, where $(V, \mathcal{E})$ is a system of linear inequalities and I is the set of implications of the form $y_1 + \cdots + y_m > 0 \Rightarrow x_1 + \cdots + x_n > 0$, $x_1, \ldots, x_n, y_1, \ldots, y_m \in V$. A *solution* to a system of linear inequalities $(V, \mathcal{E})$ is a function $S : V \to \mathbb{N}$ (or in our case $\mathbb{N}^*$) such that all inequalities are satisfied. Further, S is a solution to an enriched system of linear inequalities $(V, \mathcal{E}, I)$ if it additionally satisfies all the implications in I.

Obtaining an enriched system of linear inequalities for a KB $\mathcal{K}$ is rather straightforward. We associate a variable x_τ

to every tile $\tau \in \text{Tiles}(\mathcal{K})$ and replace every occurrence of $N(\tau)$ in the inequalities and implications in Def. 3 by x_τ. We denote the obtained system by $\mathcal{S}_\mathcal{K}$. It is easy to see that solutions of $\mathcal{S}_\mathcal{K}$ over $\mathbb{N}^*$ correspond to mosaics for $\mathcal{K}$.

4 The Translation

We briefly introduce Datalog with negation under the stable model semantics (Gelfond and Lifschitz 1988). A *(Datalog$^\neg$) program* is a set of *rules* of the form $r = h \leftarrow b_1, \ldots, b_n, not\ b_{n+1}, \ldots, not\ b_m$, where $n, m \geq 0$, $h, b_1, \ldots, b_m$ are function-free first-order atoms (referred to as atoms), and all variables in r occur in some $b_1, \ldots, b_n$. We let $head(r) = h$, $body^+(r) = \{b_1, \ldots, b_n\}$ and $body^-(p) = \{b_{n+1}, \ldots, b_m\}$. *Constraints* are rules of the form $p \leftarrow \alpha, not\ p$ (abbreviated as $\leftarrow \alpha$), where p is a fresh propositional atom. We also assume a built-in predicate $=^i$ that compares two tuples of length $i > 0$ in the obvious way. When i is clear from the context, we write $=$ instead of $=^i$. Note that these predicates can easily be axiomatized using (Datalog$^\neg$) rules. A set I of ground (variable-free) atoms is a model of a ground not-free program, if for every rule $h \leftarrow b_1, \ldots, b_n$, $b_1, \ldots, b_n \in I$ implies $h \in I$. A reduct of a program $\mathcal{P}$ w.r.t. I is given as $\mathcal{P}^I = \{head(r) \leftarrow body^+(r) : body^-(r) \cap I = \emptyset, r \in \mathcal{P}'\}$, where $\mathcal{P}'$ is the set of all ground instances of $\mathcal{P}$ over the constants in $\mathcal{P}$. We say that I is a *stable model* (or an *answer set*) of $\mathcal{P}$ if I is a $\subseteq$-minimal model of $\mathcal{P}^I$. A *Datalog$^\neg$ query* is a pair $(\mathcal{P}, Q)$, where $\mathcal{P}$ is a program and Q is a distinguished predicate. A tuple of constants $\vec{a}$ is a *certain answer* to the query $(\mathcal{P}, Q)$ over a set of atoms I if $Q(\vec{a}) \in J$, for each answer set J of $\mathcal{P} \cup I$.

For each predicate Q occurring in some TBox, we assume a predicate $\overline{Q}$ that does not occur in any TBox. For an ABox $\mathcal{A}$, we denote by $\hat{\mathcal{A}}$ the set of atoms obtained from $\mathcal{A}$ by replacing all assertions of the form $\neg Q(\vec{a})$ by $\overline{Q}(\vec{a})$. This technical trick is commonly used to accommodate negative assertions as Datalog operates on plain atoms. Note that if $\mathcal{A}$ contains no negative assertions, $\mathcal{A}$ and $\hat{A}$ coincide.

The goal of this section is to show that, given a TBox $\mathcal{T}$ and a set of predicates Σ, we can construct a Datalog$^\neg$ program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ that has the following property: for all ABoxes $\mathcal{A}$ over the signature of $\mathcal{T}$, $\mathcal{P}^{\mathcal{T},\Sigma}_{sat} \cup \hat{\mathcal{A}}$ has an answer set iff $(\mathcal{T}, \Sigma, \mathcal{A})$ is satisfiable. This program has two important properties: (i) it is polynomial in the size of $\mathcal{T}$ and Σ and (ii) it is completely independent of the data. Our translation is based on the characterization of the satisfiability problem via existence of mosaics as described in the previous section. Relying on Theorem 1, we define the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ consisting of two components $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ and $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ that communicate through a shared part of the signature. For an input ABox $\mathcal{A}$, the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sys} \cup \hat{\mathcal{A}}$ checks whether $\mathcal{A}$ respects the closed predicates and functionality assertions as specified by conditions 1. and 2. of Theorem 1, and it computes the relational representation of $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$. The program $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ together with such a representation of $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$ checks whether $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$ has solutions over $\mathbb{N}^*$. Thus, the two components together check whether $(\mathcal{T}, \Sigma, \mathcal{A})$ is satisfiable, for an input ABox $\mathcal{A}$. This is depicted in Fig. 1. It is worth noting that the $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ depends on Σ and $\mathcal{T}$ only in terms of the arities of the shared predicates and it can otherwise be used to solve arbitrary enriched systems of linear inequalities, as long as they are represented using the signature described below.

We now give a brief overview of the signature that acts as an interface between the two components. Assuming that every variable, inequality, and implication in the system has an identifier (ID), we define the following predicates for encoding enriched systems of linear inequalities:

- A unary relation Cst storing constants, including 0 and 1.
- A binary relation LEQ defining a linear order over the constants in Cst, where 0 is the least constant w.r.t. LEQ.
- Relations Var, Iq, and Im storing IDs of variables, inequalities, and implications, respectively.
- Relations $\mathsf{Iq}_{\mathsf{L}1}$ and $\mathsf{Iq}_{\mathsf{R}1}$ storing IDs of inequalities whose LHS and RHS are equal to 1, respectively.
- Relations Iq_{L} and Iq_{R} storing a pair $(\vec{q}, \vec{v})$, for each inequality ID $\vec{q}$ and a variable ID $\vec{v}$ for which the variable identified by $\vec{v}$ occurs on the left-hand side (LHS) (resp. right-hand side (RHS)) of the inequality identified by $\vec{q}$.
- Relations Im_{L} and Im_{R} storing a pair $(\vec{m}, \vec{v})$, for each implication ID $\vec{m}$ and a variable ID $\vec{v}$ for which the variable identified by $\vec{v}$ occurs on the LHS (resp. RHS) of the implication identified by $\vec{m}$.

To ease the presentation, here we focus on the intuition behind the predicates, omitting technicalities like, e.g., the arities. These will become clear in the remainder of the paper. Given an enriched system of linear inequalities $\mathcal{S}$, we use $Rel(\mathcal{S})$ to denote the relational encoding of this system.

4.1 Generating Linear Inequalities

We next show that given a TBox $\mathcal{T}$ and a set $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$, we can obtain in polynomial time the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ such that, for every ABox $\mathcal{A}$ over the signature of $\mathcal{T}$, $\mathcal{P}^{\mathcal{T},\Sigma}_{sys} \cup \hat{\mathcal{A}}$ has an answer set if and only if $\mathcal{A}$ fulfills conditions 1 and 2 in Theorem 1. More importantly, answer sets of this program correspond to $Rel(\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})})$, differing only in terms of which IDs they use for the variables, inequalities, and implications in $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$. We now sketch the construction of $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$.

The main task of $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ is to generate a relational representation of $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$, for a fixed TBox $\mathcal{T}$ and a set of predicates Σ, and any input ABox $\mathcal{A}$. As the variables in $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$ directly correspond to the tiles for $(\mathcal{T}, \Sigma, \mathcal{A})$, we first generate the tiles and then use them to build the desired enriched inequality system. Relevant dependencies among the predicates used to define $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ are depicted in Figure 2 This program consists of four main parts that given an input ABox $\mathcal{A}$ do the following:

1. Generate all possible candidate tiles for $(\mathcal{T}, \Sigma, \mathcal{A})$.
2. Eliminate the candidates that do not satisfy the conditions in Definition 2 leaving behind only proper tiles – this are the variables of $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$.
3. Generate the inequalities and implications of $\mathcal{S}_\mathcal{K}$.
4. Check whether $\mathcal{A}$ respects the closed predicates and functionality constraints.

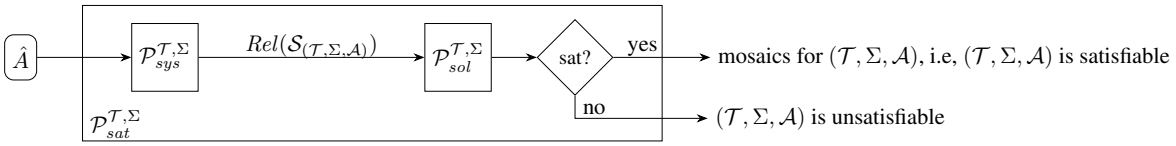


Figure 1: $\mathcal{P}^{\Sigma,\mathcal{T}}_{sat}$ and its components.

Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. The first step is to define the rules that compute the relation CandT, storing candidate tiles for $\mathcal{K}$. Like a tile, a candidate tile for $\mathcal{K}$ consists of a type T for $\mathcal{K}$ and a set of pairs (R, T'), where T' is a type for $\mathcal{K}$ and $R \subseteq \mathsf{N}^+_\mathsf{R}(\mathcal{K})$ (hereinafter referred to as role types for $\mathcal{K}$). However, the difference between the tiles and the candidate tiles is that the latter need not satisfy the conditions in Def. 2.

As types and role types are integral components of tiles, we first define the rules that compute all types and role types for $\mathcal{K}$. Recall that we have to consider an exponential number of different types and role types. As the number of the role types for $\mathcal{K}$ does not depend on the ABox, they can be encoded as usual. This means that we fix an enumeration $r_1, \ldots, r_l$ of the roles in $\mathsf{N}^+_\mathsf{R}(\mathcal{T})$ and associate to every role type R a binary string of length l that acts as an identifier for R and indicates which roles occur in R. These strings are stored in the relation RType and are computed using the rule

$$\mathsf{RType}(x_1, \ldots, x_l) \leftarrow \mathsf{Bin}(x_1), \ldots, \mathsf{Bin}(x_l),$$

where Bin is a unary relation containing constants 0 and 1. Encoding types is slightly trickier, as the number of types also depends on the ABox. Thus, the previous approach is not applicable if we wish to keep our translation data-independent, as the length of the encoding would grow with the data. We overcome this issue as follows. Similarly to before, we fix an enumeration $B_1, \ldots, B_k$ of the concept names in $\mathsf{N}_\mathsf{C}(\mathcal{T})$. We assign to every type T a string of length $k + 1$, where the first k positions are 0 or 1 indicating which concept names occur in T. Recall that there can be at most one nominal in T. The last position in the string indicates which (if any) nominal occurs in T. This position is either a constant from the KB, denoting a specific nominal, or a special constant $*$, denoting the lack of nominals in T. Thus, the relation Type contains strings $(b_1, \ldots, b_k, a)$ encoding the type $T = \{B_i : b_i = 1, 1 \leq i \leq k\} \cup \{\{a\}\}$ if $a \neq *$, or $T = \{B_i : b_i = 1, 1 \leq i \leq k\}$, otherwise. Moreover, each type is encoded by exactly one string in Type. These are computed using the rule:

$$\mathsf{Type}(x_1, \ldots, x_{k+1}) \leftarrow \mathsf{Bin}(x_1), \ldots, \mathsf{Bin}(x_k), \mathsf{Adom}(x_{k+1}),$$

where Adom is a unary relation that stores $*$ and the constants occurring in the KB, which can be extracted using a fixed number of rules. Adom together with the constants 0 and 1 makes up the relation Cst. Finally, we use Type and RType to compute the relation CandT that stores candidate tiles for $\mathcal{K}$. Let $n = |\mathcal{T}|$. Each candidate tile (T, ρ) is identified by a string of length $k + 1 + n(l + k + 1)$ such that the first $k + 1$ positions identify T, and each $l + k + 1$ positions after that identify a pair of a role type and a type in ρ. More precisely, CandT stores tuples $(\vec{p}, \vec{r}_1, \vec{p}_1 \ldots, \vec{r}_n, \vec{p}_n)$, where $\vec{p}, \vec{p}_1, \ldots, \vec{p}_n$ are in the relation Type and $\vec{r}_1, \ldots, \vec{r}_n$ are in RType. The following is the rule for computing CandT:

$$\mathsf{CandT}(\vec{x}, \vec{y}_1, \vec{x}_1, \ldots, \vec{y}_n, \vec{x}_n) \leftarrow \mathsf{Type}(\vec{x}),$$
$$\mathsf{RType}(\vec{y}_1), \mathsf{Type}(\vec{x}_1), \ldots, \mathsf{RType}(\vec{y}_n), \mathsf{Type}(\vec{x}_n)$$

It should be noted that according to Def. 2, a (candidate) tile (T, ρ) is not required to have exactly n elements in ρ. This makes the encoding via strings of fixed-length a little tricky. We overcome this issue by allowing duplicates in ρ and thus padding the candidate tiles to the desired size. However, ρ is a set and so duplicates are ignored. This means that, e.g., $(\vec{p}, \vec{r}_1, \vec{p}_1, \vec{r}_1, \vec{p}_1, \ldots, \vec{r}_1, \vec{p}_1, \vec{r}_2, \vec{p}_2)$ and $(\vec{p}, \vec{r}_1, \vec{p}_1, \vec{r}_2, \vec{p}_2, \ldots \vec{r}_2, \vec{p}_2)$, where $\vec{p}, \vec{p}_1, \vec{p}_2$ are in Type and $\vec{r}_1, \vec{r}_2$ are in RType, encode the same candidate tile. Moreover, as the order in which the elements occur in ρ is also irrelevant we get that, e.g., $(\vec{p}, \vec{r}_1, \vec{p}_1, \vec{r}_2, \vec{p}_2, \ldots, \vec{r}_n, \vec{p}_n)$ and $(\vec{p}, \vec{r}_2, \vec{p}_2, \vec{r}_1, \vec{p}_1, \ldots, \vec{r}_n, \vec{p}_n)$, for $\vec{p}, \vec{p}_1, \ldots, \vec{p}_n \in$ Type and $\vec{r}_1, \ldots, \vec{r}_n$ are in RType also encode the same candidate tile. We deal with these duplicates in the next step.

In the second step of our construction, we filter out the tuples in CandT that do not define proper tiles for $\mathcal{K}$. To this end, we go through all the conditions in Def. 2 and for each one we add the rules that "invalidate" the candidate tiles that do not satisfy this condition, i.e., they store the tuple representing this candidate tile in the relation InvTile. Defining these rules is tedious but not hard and is therefore delegated to the long version of this paper. As a small demonstration, we show here the rules that correspond to the second condition in Def. 2 (note that the first condition is satisfied by construction). For every concept inclusion $B_{i_1} \sqcap \cdots \sqcap B_{i_{m-1}} \sqsubseteq B_{i_m} \sqcup \cdots \sqcup B_{i_t}$ we have the following:

$$\mathsf{InvTile}(x_1, \ldots, x_k, \vec{y}) \leftarrow \mathsf{CandT}(x_1, \ldots, x_k, \vec{y}),$$
$$x_{i_1} = 1, \ldots, x_{i_{m-1}} = 1, x_{i_m} \neq 1, \ldots, x_{i_t} \neq 1,$$

As promised, we also address the issue that two different IDs in CandT may refer to the same candidate tile. To this end, we guess a linear order over the constants Cst using a binary predicate LEQ, where 0 is the least constant. We use the standard approach to lift this linear order to strings of length $(l+k+1)$ (see e.g., (Dantsin et al. 2001)), using the $2(l + k + 1)$-ary relation $\mathsf{LEQ}^{(l+k+1)}$. Let $\tau = (T, \rho)$ be a candidate tile. The only valid encoding of τ is a string $(\vec{p}, \vec{r}_1, \vec{p}_1, \ldots, \vec{r}_n, \vec{p}_n)$, where $\vec{p}, \vec{p}_1, \ldots, \vec{p}_n$ are in Type, $\vec{r}_1, \ldots, \vec{r}_n$ are in RType, $(\vec{r}_i, \vec{p}_i, \vec{r}_j, \vec{p}_j)$ is in LEQ^{l+k+1}, and if $(\vec{r}_i, \vec{p}_i) \neq (\vec{0}, *)$, then $(\vec{r}_i, \vec{p}_i) \neq (\vec{r}_j, \vec{p}_j)$, for all $1 \leq i, j \leq n$. All other strings encoding τ are stored using InvTile and the following rules:

$$\mathsf{InvTile}(\vec{z}) \leftarrow \mathsf{CandT}(\vec{z}), \vec{y}_i = \vec{y}_j, \vec{x}_i = \vec{x}_j, \vec{y}_i \neq (0, \ldots, 0),$$
$$\vec{x}_i \neq (0, \ldots, 0, *),$$
$$\mathsf{InvTile}(\vec{z}) \leftarrow \mathsf{CandT}(\vec{z}), \mathit{not}\ \mathsf{LEQ}^{(l+k+1)}(\vec{y}_i, \vec{x}_i, \vec{y}_j, \vec{x}_j),$$

for each $1 \leq i < j \leq n$, where $\vec{z} = (\vec{x}, \vec{y}_1, \vec{x}_1, \ldots, \vec{y}_n, \vec{x}_n)$, where $x, \vec{x}_1, \ldots, \vec{x}_n$ are vectors of length $k+1$ and $\vec{y}_1, \ldots, \vec{y}_n$ are vectors of length l.

To conclude the second step, we add the rule to compute the proper tiles for $\mathcal{K}$ and store them in the relation Tile:

$$\mathsf{Tile}(\vec{x}) \leftarrow \mathsf{CandT}(\vec{x}), \mathit{not}\ \mathsf{InvTile}(\vec{x}).$$

Thus, for every candidate tile satisfying the conditions in Def. 2, Tile contains exactly one string encoding it.

We now define the rules that build $\mathcal{S}_\mathcal{K}$. As each tile represents a variable in the system, we store tiles for $\mathcal{K}$ in the relation Var by adding the rule

$$\mathsf{Var}(\vec{x}) \leftarrow \mathsf{Tile}(\vec{x}).$$

Next, we compute the relations that encode the inequalities and implications in $\mathcal{S}_\mathcal{K}$. We assign a unique ID to each inequality and each implication in $\mathcal{S}_\mathcal{K}$, following the convention described in Table 1, and we store them in relations Iq and Im_L, respectively. Once we agree on the IDs, the rules that compute Iq and Im are straightforward. For demonstration, consider the condition M1 in Def 3. We treat each equation in M1 as two inequalities and we add the rules

$$\mathsf{Iq}(x, \vec{0}) \leftarrow \mathsf{Adom}(x), x \neq * \quad \mathsf{Iq}(\vec{0}, x) \leftarrow \mathsf{Adom}(x), x \neq *,$$

that store, for each nominal $\{c\}$, tuples $(c, \vec{0})$ and $(\vec{0}, c)$ in Iq identifying the inequalities $\sum_{\substack{(T,\rho)\in \mathrm{Tiles}(\mathcal{K}),\\ \{c\}\in T}} N((T,\rho)) \leq 1$ and $1 \leq \sum_{\substack{(T,\rho)\in \mathrm{Tiles}(\mathcal{K}),\\ \{c\}\in T}} N((T,\rho))$, respectively.

Finally, we compute the relations Iq_L, Iq_R, $\mathsf{Iq}_{\mathsf{L}1}$, $\mathsf{Iq}_{\mathsf{R}1}$, Im, and Im_R. We again demonstrate our approach on M1. On the LHS of the inequality identified by $(c, \vec{0})$ and the RHS of the inequality identified by $(\vec{0}, c)$, we have all the variables that correspond to tiles (T, ρ) for which $\{c\} \in T$. Thus, we add:

$$\mathsf{Iq}_\mathsf{L}(x, \vec{0}, \vec{x}, c, \vec{y}) \leftarrow \mathsf{Iq}(x, \vec{0}), \mathsf{Adom}(x), x \neq *, \mathsf{Var}(\vec{x}, c, \vec{y}),$$

$$\mathsf{Iq}_\mathsf{R}(\vec{0}, x, \vec{x}, c, \vec{y}) \leftarrow \mathsf{Iq}(x, \vec{0}), \mathsf{Adom}(x), x \neq *, \mathsf{Var}(\vec{x}, c, \vec{y}),$$

where $\vec{x}$ has length k.

The RHS of $(c, \vec{0})$ (resp. LHS of $(\vec{0}, c)$) is equal to 1. We encode this using the following:

$$\mathsf{Iq}_{\mathsf{R}1}(x, \vec{0}) \leftarrow \mathsf{Iq}(x, \vec{0}), \mathsf{Adom}(x), x \neq *.$$

$$\mathsf{Iq}_{\mathsf{L}1}(\vec{0}, x) \leftarrow \mathsf{Iq}(\vec{0}, x), \mathsf{Adom}(x), x \neq *.$$

The remainder of the inequalities and implications are encoded in a similar way.

We complete our construction of $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ by adding the rules that ensure that $\mathcal{A}$ respects the closed predicates and functionality restrictions. This is achieved via constraints (i.e., rules with empty bodies) and is straightforward.

4.2 Solving Linear Inequalities

We next discuss the construction of the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ that, given an enriched system of linear inequalities $\mathcal{S}$ encoded using the previously-described signature, decides whether there exists a solution over $\mathbb{N}^*$ to $\mathcal{S}$. Note that this program depends on $\mathcal{T}$ and Σ only in terms of the arity of the predicates that are shared with $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ and it can handle arbitrary enriched systems, as long as they are encoded using the given signature. If this program is run on a system generated by $\mathcal{P}^{\mathcal{T},\Sigma}_{sys} \cup \hat{\mathcal{A}}$, it decides the existence of a mosaic for $(\mathcal{T}, \Sigma, \mathcal{A})$.

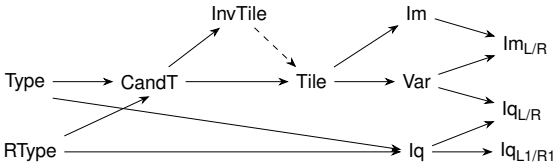


Figure 2: Partial dependency graph of $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ (negation represented via dashed arcs).

We recall a well-known result in integer programming (Papadimitriou 1981) that states that the existence of a solution over $\mathbb{N}$ implies the existence of a solution over $\mathbb{N}$ in which variables are assigned values that are at most exponential in the size of the system. This result can be generalized to solutions over $\mathbb{N}^*$ (cf. Lemma 18 in (Pratt-Hartmann 2005)) and it also holds for enriched systems of linear inequalities.

Theorem 2. *Let $(V, \mathcal{E}, I)$ be a finite enriched system of linear inequalities in which all constants and coefficients are in $\{0, \pm 1, \ldots, \pm a\}$. If $(V, \mathcal{E}, I)$ has a solution over $\mathbb{N}^*$, then it also has a solution over $\mathbb{N}^*$ where all finite values are bounded by $(|V| + |I| + |\mathcal{E}|) \cdot ((|\mathcal{E}| + |I|) \cdot a)^{2(|\mathcal{E}|+|I|)+1}$.*

Proof sketch. To decide whether an enriched system of linear inequalities $(V, \mathcal{E}, I)$ has a solution over $\mathbb{N}^*$, we construct a set of (ordinary) linear inequality systems and check whether at least one of them has a solution over $\mathbb{N}^*$. Each system in this set is of the form $(V, \mathcal{E} \cup \hat{\mathcal{E}})$, where $\hat{\mathcal{E}}$ is obtained by adding to $\mathcal{E}$ either $y_1 + \cdots + y_m \leq 0$ or $x_1 + \cdots + x_n > 0$, for each implication $y_1 + \cdots + y_m > 0 \implies x_1 + \cdots + x_n > 0$ in I. Thus, to solve these systems it is enough to consider the solutions whose finite values are bounded by $(|V| + |\hat{\mathcal{E}} \cup \mathcal{E}|) \cdot ((|\mathcal{E} \cup \hat{\mathcal{E}}|) \cdot a)^{2(|\mathcal{E} \cup \hat{\mathcal{E}}|)+1}$ (Papadimitriou 1981). The result of the theorem follows from $|\hat{\mathcal{E}}| = |I|$. □

For a given system $Rel(\mathcal{S})$, let d be the number of constants in Cst and let the arity of Var, Iq, and Im be l_v, l_e and l_i, respectively. Then, $\mathcal{S}$ has at most d^{l_v} variables, at most d^{l_e} inequalities and at most d^{l_i} implications. Let $l = (l_v + l_e + l_i)$. In view of Theorem 2, for deciding whether $\mathcal{S}$ has a solution it is sufficient to consider only those solutions whose finite values do not exceed $2^{d^{2l}}$. The maximum finite value obtainable by adding up the variables in the system is thus bounded by $2^{d^{3l}}$. These values can be encoded as binary strings of length d^{3l}, but this would make the translation exponential and also dependent on the number of constants in the data, which goes against our goal of having a polynomial, data-independent translation. We overcome this challenge in the following way: instead of having strings of length d^{3l}, we encode the addresses of these d^{3l} bits as a string of length $3l$ over the constants in Cst. We then encode the values of the variables using a $l_v + 3l + 1$-ary predicate Val, with the following meaning: $(\vec{x}, \vec{z}, b)$ in the relation Val denotes that

Cond. in Def 3	Factors	IDs
M1	$\{c\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$	$(c, \vec{0})$ and $(\vec{0}, c)$,where $\vec{0}$ is a 0-vector of length $2k+l+1$.
M2	$-$	$(\vec{0})$, where $\vec{0}$ is a 0-vector of length $2(k+1)+l$.
M3	$T, T' \in \text{Types}(\mathcal{K})$ and $R \subseteq \mathsf{N}_\mathsf{R}^+(\mathcal{K})$	(t, t', r), where t, t' identify T and T', respectively, and r identifies R.
M4	$\tau = (T, \rho) \in \text{Tiles}(\mathcal{K})$ and $(R, T') \in \rho$	(t, r, t'), where t is the ID of τ, r identifies R, and t' identifies T'.
M5	$\{a\}, \{b\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$ and $B_i, B_j \in \mathsf{N}_\mathsf{C}(\mathcal{K})$	$(a, b, \vec{b}, \vec{b}', \vec{0})$, where $\vec{b}$ (resp. $\vec{b}'$) is of length k s.t. the i-th (resp. j-th) position is 1 and the rest are 0, and $\vec{0}$ is a 0-vector of length $n(k+1+l)+l$.

Table 1: Identifiers of inequalities and implications of $\mathcal{S}_\mathcal{K}$.

the bit encoded by the $\vec{z}$ in the value of the variable encoded by $\vec{x}$ has the value b, where b is either 0 or 1.

We are ready to present our construction. The program $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ guesses the value of each variable and checks whether the guess is a valid solution to the system. The guessing part is rather straightforward. In the remainder of this section, we write $\mathsf{Cst}^i(x_1, \ldots, x_i)$ to abbreviate $\mathsf{Cst}(x_1), \ldots \mathsf{Cst}(x_i)$, for $i \geq 1$. We begin by adding the rules for guessing which variables are set to infinity:

$$\mathsf{Fin}(\vec{x}) \leftarrow \mathsf{Var}(\vec{x}), \mathit{not}\ \mathsf{Inf}(\vec{x}) \quad \mathsf{Inf}(\vec{x}) \leftarrow \mathsf{Var}(\vec{x}), \mathit{not}\ \mathsf{Fin}(\vec{x})$$

If a variable is not set to infinity, we guess its value bit by bit using the following rules:

$$\mathsf{Val}(\vec{x}, \vec{z}', \vec{z}, 0) \leftarrow \mathsf{Var}(\vec{x}), \mathsf{Fin}(\vec{x}), \mathsf{Cst}^l(\vec{z}'), \mathsf{Cst}^{2l}(\vec{z}), \vec{z}' \neq (0, \ldots, 0)$$

$$\mathsf{Val}(\vec{x}, \vec{z}, 0) \leftarrow \mathsf{Var}(\vec{x}), \mathsf{Fin}(\vec{x}), \mathsf{Cst}^{3l}(\vec{z}), \mathit{not}\ \mathsf{Val}(\vec{x}, \vec{z}, 1)$$

$$\mathsf{Val}(\vec{x}, \vec{z}, 1) \leftarrow \mathsf{Var}(\vec{x}), \mathsf{Fin}(\vec{x}), \mathsf{Cst}^{3l}(\vec{z}), \mathit{not}\ \mathsf{Val}(\vec{x}, \vec{z}, 0)$$

As the variables take the values bounded by $2^{d^{2l}}$, we only need the first d^{2l} bits to encode them. The remainder of the d^l bits have their value set to 0 and are reserved for accommodating addition. The first rule thus sets the value of the last d^l bits to 0. The other two rules freely guess the values of the remaining bits.

We now move to the checking part of $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$. We use the l_e-ary predicate Sat to store which inequalities are satisfied by our guess. Note that if a variable that is set to infinity occurs on the RHS of some inequality, that inequality is automatically satisfied. Thus, we add the following rule:

$$\mathsf{Sat}(\vec{y}) \leftarrow \mathsf{Var}(\vec{x}), \mathsf{Iq}(\vec{y}), \mathsf{Iq}_\mathsf{R}(\vec{y}, \vec{x}), \mathsf{Inf}(\vec{x})$$

Further, we mark which inequalities have occurrences of variables that are set to infinity using the predicate InfIq:

$$\mathsf{InfIq}(\vec{y}) \leftarrow \mathsf{Iq}(\vec{y}), \mathsf{Var}(\vec{x}), \mathsf{Inf}(\vec{x}), \mathsf{Iq}_\mathsf{Q}(\vec{y}, \vec{x}),$$

where $\mathsf{Q} \in \{\mathsf{L}, \mathsf{R}\}$. We store the remaining inequalities in the relation FinIq computed as:

$$\mathsf{FinIq}(\vec{y}) \leftarrow \mathsf{Iq}(\vec{y}), \mathit{not}\ \mathsf{InfIq}(\vec{y}).$$

To check the satisfaction of the inequalities in which only the variables with finite values occur we do the following. For each such inequality, we compute the value on its LHS (resp. RHS) incrementally, by iterating through the variables and at each iteration storing the sum of all the variables considered so far that occur on the LHS (resp. RHS). To this end, we define a linear order over the variables. We use the linear order from the relation LEQ and we lift it to strings over Cst of length at most $3l$. We further extract the relations $\mathsf{First}^i, \mathsf{Last}^i, \mathsf{Succ}^i$, $1 \leq i \leq 3l$, that store the least string of length i, the greatest string of length i, and the successor relation on the strings of length i, respectively. Finally, not every string over Cst represents a variable in $\mathcal{S}$. Thus, we use the linear order above to define a successor relation on the variables in the $2l_v$-ary relation Succ_V. Relations $\mathsf{First}_\mathsf{V}$ and Last_V store, respectively, the first and the last variable with respect to this successor relation.

We store the intermediate results using a $(l_e + l_v + 3l + 1)$-ary predicate Upto_L. Intuitively, $(\vec{q}, \vec{v}, \vec{p}, b)$ is in Upto_L if, for the LHS of inequality $\vec{q}$, the $\vec{p}$-th bit in the sum of variables up to $\vec{v}$ (including $\vec{v}$) has the value b. We do this until we reach the last variable. Next, we add the facts and rules that define bit-by-bit addition of binary numbers. To this end, we use a 5-ary relation Add, where a tuple (b, b', b'', c, r) in Add denotes that the result of adding bits b, b', and b'' is r with the carry c. As these rules are standard, we omit them here. Further, we use the $(l_e + l_v + 3l + 1)$-ary predicate $\mathsf{Carry}_\mathsf{L}$ to mark relevant carry bits. A tuple $(\vec{q}, \vec{v}, \vec{p}, c)$ is in $\mathsf{Carry}_\mathsf{L}$ if when adding up the bit at the position $\vec{p}$ of the variable $\vec{x}$ and the result so far, we need to take into account a a bit with the value c that was carried over from the previous computation.

We are now ready to compute LHS of the inequalities (RHS defined analogously) and store our results using the $l_e + 3l + 1$-ary predicate LHS. To deal with the case when we know that the LHS is equal to 1, we have:

$$\mathsf{LHS}(\vec{y}, \vec{z}, 1) \leftarrow \mathsf{FinIq}(\vec{y}), \mathsf{Iq}_{\mathsf{L}1}(\vec{y}), \mathsf{First}^3 l(\vec{z})$$

$$\mathsf{LHS}(\vec{y}, \vec{z}, 0) \leftarrow \mathsf{FinIq}(\vec{y}), \mathsf{Iq}_{\mathsf{L}1}(\vec{y}), \mathsf{Cst}^{3l}(\vec{z}), \mathit{not}\ \mathsf{First}^3 l(\vec{z})$$

We then add the remaining rules:

$$\mathsf{Upto}_\mathsf{L}(\vec{y}, \vec{x}, \vec{z}, x) \leftarrow \mathsf{FinIq}(\vec{y}), \mathsf{First}_\mathsf{V}(\vec{x}), \mathsf{Val}(\vec{x}, \vec{z}, x), \mathit{not}\ \mathsf{Iq}_{\mathsf{L}1}(\vec{y})$$

$$\mathsf{Carry}_\mathsf{L}(\vec{y}, \vec{x}, \vec{z}, 0) \leftarrow \mathsf{FinIq}(\vec{y}), \mathsf{First}_\mathsf{V}(\vec{x}), \mathsf{Cst}^{3l}(\vec{z})$$

$$\mathsf{Carry}_\mathsf{L}(\vec{y}, \vec{x}, \vec{z}, 0) \leftarrow \mathsf{FinIq}(\vec{y}), \mathsf{Var}(\vec{x}), \mathsf{First}^{3l}(\vec{z})$$

$$\mathsf{Upto}_\mathsf{L}(\vec{y}, \vec{x}, \vec{z}, x') \leftarrow \mathsf{Upto}_\mathsf{L}(\vec{y}, \vec{x}', \vec{z}, x'), \mathsf{Succ}_\mathsf{V}(\vec{x}', \vec{x}), \mathit{not}\ \mathsf{Iq}_\mathsf{L}(\vec{y}, \vec{x}),$$

$$\mathsf{Upto_L}(\vec{y},\vec{x},\vec{z},x) \leftarrow \mathsf{Upto_L}(\vec{y},\vec{x}',\vec{z},x'), \mathsf{Succ_V}(\vec{x}',\vec{x}), \mathsf{Iq_L}(\vec{y},\vec{x}), \mathsf{Val}(\vec{x},\vec{z},x'') \mathsf{Carry_L}(\vec{y},\vec{x},\vec{z},z), \mathsf{Add}(x',x'',z,y,x)$$

$$\mathsf{Carry_L}(\vec{y},\vec{x},\vec{z}',y) \leftarrow \mathsf{Upto_L}(\vec{y},\vec{x}',\vec{z},x'), \mathsf{Succ_V}(\vec{x}',\vec{x}), \mathsf{Iq_L}(\vec{y},\vec{x}), \mathsf{Val}(\vec{x},\vec{z},x''), \mathsf{Succ}^{3l}(\vec{z},\vec{z}') \mathsf{Carry_L}(\vec{y},\vec{x},\vec{z},z), \mathsf{Add}(x',x'',z,y,x),$$

$$\mathsf{LHS}(\vec{y},\vec{z},b) \leftarrow \mathsf{Upto_L}(\vec{y},\vec{x},\vec{z},b), \mathsf{Last_V}(\vec{x})$$

We next make sure the inequalities are satisfied.

$$\mathsf{Sat}(\vec{y}) \leftarrow \mathsf{LHS}(\vec{y},\vec{z},c), \mathsf{RHS}(\vec{y},\vec{z},c'), \mathsf{Last}^{3l}(\vec{z}), \mathsf{LEQ}(c,c'), c \neq c'$$

$$\mathsf{Sat}'(\vec{y},\vec{z}) \leftarrow \mathsf{LHS}(\vec{y},\vec{z},c), \mathsf{RHS}(\vec{y},\vec{z},c'), \mathsf{Last}^{3l}(\vec{z}), c = c'$$

$$\mathsf{Sat}(\vec{y}) \leftarrow \mathsf{LHS}(\vec{y},\vec{z}',c), \mathsf{RHS}(\vec{y},\vec{z}',c'), \mathsf{Sat}'(\vec{y},\vec{z}) \mathsf{Succ}^{3l}(\vec{z}',\vec{z}), \mathsf{LEQ}(c,c'), c \neq c'$$

$$\mathsf{Sat}'(\vec{y},\vec{z}) \leftarrow \mathsf{Sat}'(\vec{y},\vec{z}'), \mathsf{Succ}^{3l}(\vec{z},\vec{z}'), \mathsf{LHS}(\vec{y},\vec{z},c), \mathsf{RHS}(\vec{y},\vec{z},c'), c = c'$$

$$\mathsf{Sat}(\vec{y}) \leftarrow \mathsf{Sat}'(\vec{y},\vec{z}), \mathsf{First}^{3l}(\vec{z}) \qquad \leftarrow \mathsf{Iq}(\vec{y}), \mathit{not}\ \mathsf{Sat}(\vec{y})$$

Note that we do not specifically deal with the case where variables that are set to infinity occur on the LHS but not on the RHS of some inequality. Such an inequality $\vec{y}$ is not be satisfied and indeed, due to the stable model semantics, we are not able to derive $\mathsf{Sat}(\vec{y})$.

To finish the construction, we add the rules ensuring that all the implications are satisfied. For $\mathsf{Q} \in \{\mathsf{L},\mathsf{R}\}$, we add:

$$\mathsf{GT0_Q}(\vec{y}) \leftarrow \mathsf{Im_Q}(\vec{y},\vec{x}), \mathsf{Cst}^{3l}(\vec{z}), \mathit{not}\ \mathsf{Val}(\vec{x},\vec{z},0)$$
$$\leftarrow \mathsf{Im}(\vec{y}), \mathsf{GT0_L}(\vec{y}), \mathit{not}\ \mathsf{GT0_R}(\vec{y})$$

Proposition 1. *$\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ is polynomial in the size of $\mathcal{T}$ and Σ and $\mathcal{S}$ has a solution over $\mathbb{N}^*$ iff $\mathcal{P}^{\mathcal{T},\Sigma}_{sol} \cup Rel(\mathcal{S})$ has an answer set.*

From the results in the previous section and Proposition 1 we obtain the result below.

Theorem 3. *For a TBox $\mathcal{T}$ and $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$, we can obtain a program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ in polynomial time such that $\mathcal{P}^{\mathcal{T},\Sigma}_{sat} \cup \hat{\mathcal{A}}$ has an answer set if and only if $(\mathcal{T},\Sigma,\mathcal{A})$ is satisfiable, for all ABoxes $\mathcal{A}$ over the signature of $\mathcal{T}$.*

5 Query Answering and Complexity

In this section, we introduce the notion of ontology-mediated queries with closed predicates and show how we can answer them using Datalog with negation.

Let $\mathsf{N_V}$ be a countably infinite set of variables. An *ontology-mediated query (with closed predicates)* (OMQ) is a triple $Q = (\mathcal{T},\Sigma,q)$, where $\mathcal{T}$ is a TBox, $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ is a set of closed predicates and q is an ordinary first-order (FO) query over the predicates in $\mathsf{N_C} \cup \mathsf{N_R}$, the constants in $\mathsf{N_I}$ and the variables from $\mathsf{N_V}$. A *match* π for q in an interpretation $\mathcal{I}$ is a function that maps every constant to itself and every free variable of q onto an element of $\Delta^{\mathcal{I}}$ such that the query obtained from q by substituting $\pi(x)$ for each free variable x in q is true in $\mathcal{I}$. Let $Q = (\mathcal{T},\Sigma,q)$ be an OMQ with $x_1,\ldots,x_n$ being the free variables of q. A tuple of constants $(a_1,\ldots,a_n)$ is a *certain answer* to Q over an ABox $\mathcal{A}$, if in every $\mathcal{I}$ with $\mathcal{I} \models (\mathcal{T},\Sigma,\mathcal{A})$, there exists a match π for q such that $\pi(x_i) = a_i$, for all $i = 1,\ldots,n$. The *query answering problem* is the problem of deciding, given an OMQ Q, an ABox $\mathcal{A}$, and a tuple of constants $\vec{a}$, whether $\vec{a}$ is a certain answer to Q over $\mathcal{A}$.

We first focus on ontology-mediated *instance queries* (IQs) and then lift our approach to cover a larger class of OMQs.

Instance Queries. Instance queries are OMQs of the form $Q = (\mathcal{T},\Sigma,P(\vec{x}))$, where $P \in \mathsf{N_C} \cup \mathsf{N_R}$ and $\vec{x} \in \mathsf{N_V}$ if $P \in \mathsf{N_C}$, otherwise $\vec{x} \in \mathsf{N_V}^2$. Given a KB $\mathcal{K} = (\mathcal{T},\Sigma,\mathcal{A})$, we say that an ABox $\mathcal{A}' \subseteq \mathcal{A}$ is a *completion of $\mathcal{A}$ w.r.t $\mathcal{T}$ and Σ* if the following holds: (i) for each $a \in \mathsf{N_I}(\mathcal{K})$ and concept name $C \in \mathsf{N_C}(\mathcal{K})$, either $C(a) \in \mathcal{A}'$ or $\neg C(a) \in \mathcal{A}'$, (ii) for each $a,b \in \mathsf{N_I}(\mathcal{K})$ and role name $r \in \mathsf{N_R}(\mathcal{K})$, either $r(a,b) \in \mathcal{A}'$ or $\neg r(a,b) \in \mathcal{A}'$, (iii) $C(a) \in \mathcal{A}'$ and $C \in \Sigma$ implies $C(a) \in A$, and (iv) $r(a,b) \in \mathcal{A}'$ and $r \in \Sigma$ implies $r(a,b) \in \mathcal{A}$. We say that $\mathcal{A}'$ is a *consistent completion of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ* if $\mathcal{A}'$ is a completion of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ, and $(\mathcal{T},\Sigma,\mathcal{A}')$ is satisfiable. Instance queries have a convenient property: a tuple of constants $\vec{a}$ is a certain answer to the IQ $(\mathcal{T},\Sigma,P(\vec{x}))$ over $\mathcal{A}$ iff $P(\vec{a}) \in \mathcal{A}'$, for every consistent completion $\mathcal{A}'$ of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ.

Let $Q = (\mathcal{T},\Sigma,P(\vec{x}))$ be an IQ. Relying on this property, we show how to obtain a program $\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ}$ such that the certain answers to Q over $\mathcal{A}$ coincide with the certain answers to the Datalog$^{\neg}$ query $(\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ},P)$ over $\hat{\mathcal{A}}$, for all ABoxes $\mathcal{A}$ over the signature of $\mathcal{T}$. Consider the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ from the previous section. Even though each answer set of $\mathcal{P}^{\mathcal{T},\Sigma}_{sat} \cup \hat{\mathcal{A}}$ implicitly corresponds to a consistent completion of $\mathcal{A}$ and vice versa, due to the stable model semantics, this program does not infer any new atoms over the signature of the TBox and can therefore not be directly used for query answering. Thus, we add, for all $A \in \mathsf{N_C}(\mathcal{T}) \setminus \Sigma$, $B \in \mathsf{N_C}(\mathcal{T})$, and $r \in \mathsf{N_R}(\mathcal{T}) \setminus \Sigma$, the rules that nondeterministically guess a completion of an input ABox $\mathcal{A}$ over the signature of $\mathcal{T}$:

$$A(x) \leftarrow \mathsf{Adom}(x), x \neq *, \mathit{not}\ \overline{A}(x)$$
$$\overline{B}(x) \leftarrow \mathsf{Adom}(x), x \neq *, \mathit{not}\ B(x)$$
$$r(x,y) \leftarrow \mathsf{Adom}(x), \mathsf{Adom}(y), \mathit{not}\ \overline{r}(x,y), x \neq *, y \neq *$$
$$\overline{r}(x,y) \leftarrow \mathsf{Adom}(x), \mathsf{Adom}(y), \mathit{not}\ r(x,y), x \neq *, y \neq *$$

It is easy to see that $\mathcal{A}'$ is a consistent completion of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ iff there is an answer set I of $\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ}$ with $\hat{\mathcal{A}}' \subseteq I$. As $\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ}$ is obtained from $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ in polynomial time, which is in turn obtained in polynomial time from $\mathcal{T}$ and Σ, we have the following result:

Theorem 4. *Let $Q = (\mathcal{T},\Sigma,P(\vec{x}))$ be an IQ. We can obtain in polynomial time a program $\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ}$ such that the certain answers to Q over $\mathcal{A}$ coincide with the certain answers of $(\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ},P)$ over $\hat{\mathcal{A}}$, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$.*

Theorem 5. *IQs mediated by $\mathcal{ALCHOIF}$ ontologies with closed predicates are co-*NP*-complete in data complexity.*

$$\bullet\ rr(R(t_1,\ldots,t_n)) := \begin{cases} \text{variables in } t_1,\ldots,t_n, & \text{if } R \in \Sigma \\ \emptyset, & \text{otherwise} \end{cases} \qquad \bullet\ rr(x=a) = rr(a=x) := \{x\}$$

$$\bullet\ rr(\varphi_1 \wedge \varphi_2) := rr(\varphi_1) \cup rr(\varphi_2) \qquad \bullet\ rr(\varphi_1 \vee \varphi_2) := rr(\varphi_1) \cap rr(\varphi_2) \qquad \bullet\ rr(\neg\psi) := \emptyset$$

$$\bullet\ rr(\psi \wedge x=y) := \begin{cases} rr(\psi), & \text{if } \{x,y\} \cap rr(\psi) = \emptyset \\ rr(\psi) \cup \{x,y\}, & \text{otherwise} \end{cases} \qquad \bullet\ rr(\exists x\psi) := \begin{cases} rr(\psi) \setminus \{x\}, & \text{if } x \in rr(\psi) \\ \text{fail}, & \text{otherwise} \end{cases}$$

Table 2: Computation of Σ-range-restricted variables in φ.

Proof sketch. Checking whether a tuple of constants is a certain answer to a Datalog$^\neg$ query is co-NP-complete in terms of data complexity (Dantsin et al. 2001). As the obtained query does not depend on $\mathcal{A}$, and $\hat{\mathcal{A}}$ is obtained from $\mathcal{A}$ in polynomial time, we get the desired upper data complexity bound. The matching lower bound comes the fact that IQs are co-NP-complete in data complexity for $\mathcal{ALC}$ even without closed predicates (Schaerf 1993). □

Safe-Range Queries. We now extend our results to a larger class of OMQs. Inspired by safe-range FO queries used in relational databases for ensuring domain independence (Abiteboul, Hull, and Vianu 1995), we define safe-range OMQs $Q = (\mathcal{T}, \Sigma, \varphi)$ where φ is an FO query in which each variable ranges only over the constants occurring in the predicates from Σ (i.e., the closed predicates) in the ABox over which Q is being answered. To formally define these queries, we introduce the notion of Σ-*safe-range* FO queries, for a given set of predicates Σ. Checking whether an FO query φ is Σ-safe-range can is done syntactically and consists of the following steps. First, φ is transformed into an equivalent query φ' in *safe-range normal form* (SRNF) (see e.g., (Abiteboul, Hull, and Vianu 1995)). Then the set of Σ-range-restricted variables in φ' is computed using the rules in Table 2. If at any point we obtain "fail", φ' is not Σ-safe-range. Otherwise, we check if every free variable of φ' is Σ-safe-range. If this is the case, then φ' is range-restricted, otherwise it is not. Finally, we define *safe-range OMQs* as OMQs $Q = (\mathcal{T}, \Sigma, \varphi)$ in which φ is Σ-safe-range FO formula.

Proposition 2. *Given a tuple of constants $\vec{a} = (a_1,\ldots,a_n)$ and a safe-range OMQ $Q = (\mathcal{T}, \Sigma, \varphi)$, where $x_1,\ldots,x_n$ are free variables of φ, $\vec{a}$ is a certain answer to Q over $\mathcal{A}$ iff $\mathcal{A}' \models \varphi[a_1/x_1,\ldots,a_n/x_n]$, for every consistent completion of $\mathcal{A}$ w.r.t. $(\mathcal{T}, \Sigma, \mathcal{A})$.*

Every Σ-safe-range FO query q as a Datalog$^\neg$ program r_q that is polynomial in the size of q and that, given a set of ground atoms I, computes the answers to q over I and stores them in the relation P_q. This can be done by simply using the rewriting procedure from the literature defined for ordinary safe-range FO queries. We note that the program r_q uses only stratified negation, and so it has exactly one answer set (Apt, Blair, and Walker 1988). Thus, we have that $\vec{a}$ is an answer to q over I iff $P_q(\vec{a})$ occurs in the answer set of $r_q \cup I$.

Theorem 6. *Let $Q = (\mathcal{T}, \Sigma, q)$ be a safe-range OMQ. We can obtain in polynomial time a program $\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ}$ from $\mathcal{T}$ and Σ such that the certain answers to Q over $\mathcal{A}$ coincide with the certain answers of $(\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ} \cup r_q, P_q)$ over $\hat{\mathcal{A}}$, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$.*

Theorem 7. *Answering safe-range OMQs is co-*NP*-complete in data complexity for $\mathcal{ALCHOIF}$.*

6 Discussion

In this paper, we presented a translation of $\mathcal{ALCHOIF}$ with closed predicates into Datalog with stable negation. Our translation uses a very different approach from the other translations in the literature and it is based on a characterization of the satisfiability problem for this logic as a system of linear inequalities with some side conditions. Given a TBox $\mathcal{T}$ and a set of closed predicates Σ, we had first shown how to construct in polynomial time a program that takes as an input an ABox $\mathcal{A}$ and decides whether the knowledge base $(\mathcal{T}, \Sigma, \mathcal{A})$ is satisfiable. We then showed how to further extend this program to answer instance queries and safe-range OMQs. As a by-product of our translation we obtained a proof that these queries are co-NP-complete in data complexity.

In the future we would like to extend this approach to $\mathcal{ALCHOIQ}$ as well as to investigate the *absolute expressive power* of OMQs with $\mathcal{ALCHOIF}$ ontologies and closed predicates, i.e., we would like to know if the considered query languages are powerful enough to capture all database queries computable in co-NP. Our approach already covers a very large class of OMQs, namely the safe-range OMQs. Going above this class would be difficult as it is known that first-order queries quickly become undecidable, even for very basic extensions of conjunctive queries (CQs) and very lightweight DLs (Gutiérrez-Basulto et al. 2015). Also considering conjunctive queries is not really a viable option, since it is known that $\mathcal{ALCOIF}$ mediated CQs are co-N2ExpTime-hard even in the absence of closed predicates (Glimm, Kazakov, and Lutz 2011). Thus, due to computational complexity reasons, there cannot exist a polynomial translation of $\mathcal{ALCHOIF}$ mediated CQs into Datalog with stable negation. As a final remark, we note that the considered variant of Datalog underlies Answer Set Programming (ASP), which is a very mature area, and many efficient reasoning engines for this rule language exist. While our polynomial time translation is unlikely to yield an efficient tool for reasoning with $\mathcal{ALCHOIF}$ ontologies, it nevertheless draws an important new connection between DLs and ASP.

Acknowledgments

This work was supported by the Vienna Business Agency, and the Austrian Science Fund (FWF) projects P30360, P30873 and W1255.

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Ahmetaj, S.; Ortiz, M.; and Šimkus, M. 2020. Polynomial rewritings from expressive description logics with closed predicates to variants of datalog. *Artif. Intell.* 280:103220.

Apt, K. R.; Blair, H. A.; and Walker, A. 1988. Towards a theory of declarative knowledge. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann. 89–148.

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Benedikt, M.; Bourhis, P.; ten Cate, B.; and Puppis, G. 2016. Querying visible and invisible information. In *Proc. of LICS 2016*, 297–306. ACM.

Bienvenu, M.; ten Cate, B.; Lutz, C.; and Wolter, F. 2014. Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.* 39(4):33:1–33:44.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning* 39(3):385–429.

Calvanese, D. 1996. Finite model reasoning in description logics. In Aiello, L. C.; Doyle, J.; and Shapiro, S. C., eds., *Proc. of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, 292–303. Morgan Kaufmann.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33(3):374–425.

Franconi, E.; Ibáñez-García, Y. A.; and Seylan, I. 2011. Query answering with DBoxes is hard. *Electr. Notes Theor. Comput. Sci.* 278:71–84.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of ICLP/SLP 1988*, 1070–1080. MIT Press.

Glimm, B.; Kazakov, Y.; and Lutz, C. 2011. Status QIO: an update. In Rosati, R.; Rudolph, S.; and Zakharyaschev, M., eds., *Proc. of the 24th International Workshop on Description Logics (DL 2011)*, volume 745 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Gogacz, T.; Gutiérrez-Basulto, V.; Ibáñez-García, Y. A.; Murlak, F.; Ortiz, M.; and Šimkus, M. 2020. Ontology focusing: Knowledge-enriched databases on demand. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*. IOS Press. Extended paper available at http://arxiv.org/abs/1904.00195.

Gottlob, G., and Schwentick, T. 2012. Rewriting ontological queries into small nonrecursive datalog programs. In *Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012)*. AAAI Press.

Gutiérrez-Basulto, V.; Ibáñez-García, Y.; Kontchakov, R.; and Kostylev, E. V. 2015. Queries with negation and inequalities over lightweight ontologies. *Journal of Web Semantics* 35:184–202.

Hustadt, U.; Motik, B.; and Sattler, U. 2007. Reasoning in description logics by a reduction to disjunctive datalog. *J. Autom. Reasoning* 39(3):351–384.

Lutz, C.; Sattler, U.; and Tendera, L. 2005. The complexity of finite model reasoning in description logics. *Inf. Comput.* 199(1-2):132–171.

Lutz, C.; Seylan, I.; and Wolter, F. 2013. Ontology-based data access with closed predicates is inherently intractable(sometimes). In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing (IJCAI 2013)*, 1024–1030. IJCAI/AAAI.

Ortiz, M.; Rudolph, S.; and Šimkus, M. 2010. Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In *Proc. of the 12th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010)*. AAAI Press.

Papadimitriou, C. H. 1981. On the complexity of integer programming. *Journal of the ACM (JACM)* 28(4):765–768.

Pratt-Hartmann, I. 2005. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information* 14(3):369–395.

Schaerf, A. 1993. On the complexity of the instance checking problem in concept languages with existential quantification. *J. Intell. Inf. Syst.* 2(3):265–278.

Tobies, S. 2000. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *J. of Artificial Intelligence Research* 12:199–217.