Ontology-Mediated Querying on Databases of Bounded Cliquewidth

Carsten Lutz\textsuperscript{1}, Leif Sabellek\textsuperscript{2}, Lukas Schulze\textsuperscript{1}

\textsuperscript{1}Department of Computer Science, Leipzig University, Germany\textsuperscript{2}Department of Computer Science, University of Bremen, Germany

\{clu, lschulze\}@informatik.uni-leipzig.de, sabellek@uni-bremen.de

Abstract

We study the evaluation of ontology-mediated queries (OMQs) on databases of bounded cliquewidth from the viewpoint of parameterized complexity theory. As the ontology language, we consider the description logics \textit{ALC} and \textit{ALCI} as well as the guarded two-variable fragment \textit{GF\textsubscript{2}} of first-order logic. Queries are atomic queries (AQs), conjunctive queries (CQs), and unions of CQs. All studied OMQ problems are fixed-parameter linear (FPL) when the parameter is the size of the OMQ plus the cliquewidth. Our main contribution is a detailed analysis of the dependence of the running time on the parameter, exhibiting several interesting effects.

1 Introduction

Ontology-mediated querying is an established subfield of knowledge representation. The general aim is to enrich a database with an ontology to add domain knowledge and to extend the vocabulary available for query formulation (Bienvenu et al. 2014; Bienvenu and Ortiz 2015; Calvanese et al. 2009). Expressive description logics (DLs) such as \textit{ALC} and \textit{ALCI} are a popular choice for the ontology language as they underpin the widely used OWL DL profile of the OWL 2 ontology language. However, ontology-mediated querying with these languages is \textit{coNP}-complete in data complexity even for simple atomic queries (AQs) of the form \textit{A}(x) and thus scales poorly to larger amounts of data. One reaction to this problem is to resort to approximate query answers, as done for example in (Zhou et al. 2015; Haga et al. 2021). A potential alternative is to only admit databases from a class that is sufficiently restricted so that non-approximate answers can be computed in \textsc{Ptime}, or to decompose the input database into component databases from such a class.

The aim of this paper is to study ontology-mediated querying on classes of databases of bounded cliquewidth. This is relevant because such classes are maximal known ones on which ontology-mediated querying with DLs such as \textit{ALC} and \textit{ALCI} is in \textsc{Ptime} in data complexity, even when unions of conjunctive queries (UCQs) are admitted as queries. This follows from a folklore translation of ontology-mediated querying into \textsc{MSO\textsubscript{1}} and the known result that \textsc{MSO\textsubscript{1}} has \textsc{Ptime} data complexity on structures of bounded cliquewidth (Courcelle, Makowsky, and Rotics 2000). In fact, it even has linear time data complexity (when a \textit{k}-expression for constructing the database is provided). We recall that bounded treewidth implies bounded cliquewidth since every database of treewidth \textit{k} has cliquewidth at most \(3 \cdot 2^k - 1\) (Corneil and Rotics 2005). The converse is false because databases of bounded treewidth are sparse in the sense of graph theory, whereas databases of bounded cliquewidth may be dense.

We do not claim that databases encountered in practical applications typically have small cliquewidth. In fact, an empirical study of real-world databases has found that their treewidth tends to be high (Maniu, Senellart, and Jog 2019), and one may expect similar results for cliquewidth. The same study, however, also points out the opportunities that lie in the decomposition of a database into a part of high treewidth and parts of low treewidth, and this has in fact been used successfully to implement efficient querying (Wei 2010; Akiba, Sommer, and Kawarabayashi 2012; Maniu, Cheng, and Senellart 2017). While we are not aware that similar approaches based on cliquewidth have yet been studied, it seems entirely reasonable to pursue them. The work presented here may provide a foundation for such an endeavour.

In this paper, we focus on the framework of parameterized complexity theory. Linear time data complexity with a uniform algorithm trivially implies \textit{fixed-parameter linearity} (FPL) when the parameter is the size of the ontology-mediated query (OMQ). Recall that FPL is defined like the more familiar \textit{fixed-parameter tractability} (FPT) except that the running time may depend only linearly on the size of the database, that is, it must be \(f(|Q|) \cdot |D|\) where \(f\) is a computable function, \(Q\) the OMQ, and \(D\) the database. Our main goal is to determine the optimal running time of FPL algorithms for ontology-mediated querying on databases of bounded cliquewidth and to provide algorithms that achieve this running time.

As the ontology language, we consider the DLs \textit{ALC} and \textit{ALCI} and the more expressive guarded two-variable fragment \textit{GF\textsubscript{2}} of first-order logic. As queries, we admit AQs, conjunctive queries (CQs), and UCQs. An \textit{ontology-mediated query (OMQ)} combines an ontology and a query, and an \textit{OMQ language} is determined by the choice of an ontology language and a query language. For instance, \{\textit{ALC}, \textit{AQ}\} is the OMQ language that combines \textit{ALC} ontologies with AQs. We find that the running time may be single or double exponential in \(|Q|\), depending on the choice of the OMQ language. There are several surprising effects, which we highlight in the following. An overview is provided in Table 1.
We show that in \((\mathcal{ACCI}, \mathcal{AQ})\), evaluating an OMQ \(Q\) on a database \(D\) is possible in time \(2^{O(|Q| \cdot k^2)} \cdot |D|\) where \(k\) is the cliquewidth of \(D\).\(^1\) We assume here that a \(k\)-expression for constructing \(D\) is given (otherwise, the running time is cubic in \(|D|\) and thus FPT, but not FPL). Note that this implies FPL even when the parameter is the size of the OMQ plus the cliquewidth of the database (and so the cliquewidth of the database needs not be bounded by a constant). In \((\mathcal{GF}_2, \mathcal{AQ})\), in contrast, we achieve a running time of \(2^{O(|Q|\cdot k^2)} \cdot |D|\) and show that attaining \(2^{2^{|Q|}} \cdot \text{poly}(|D|)\) is impossible even on databases of cliquewidth 2 unless the exponential time hypothesis (ETH) fails. This is interesting for several reasons. First, it is folklore that OMQ evaluation in \((\mathcal{GF}_2, \mathcal{AQ})\) is \textsc{ExpTime}-complete in combined complexity on unrestricted databases, and thus one has to ‘pay’ for the running time to be polynomial in \(|D|\) on databases of bounded cliquewidth by an exponential increase in the running time in \(|Q|\). Second, the higher running time for \((\mathcal{GF}_2, \mathcal{AQ})\) compared to \((\mathcal{ACCI}, \mathcal{AQ})\) is not due to the fact that \(\mathcal{GF}_2\) is more expressive than \(\mathcal{ACCI}\). In fact, the \(\mathcal{GF}_2\) ontology used in the lower bound proof can be expressed in \(\mathcal{ACCI}\) and the increase in complexity is due to the higher succinctness of \(\mathcal{GF}_2\). And third, we also show that in \((\mathcal{GF}_2, \mathcal{AQ})\), OMQs can be evaluated in time \(2^{O(|Q|^2 \cdot k^2)} \cdot |D|\) where \(k\) is the treewidth of \(D\). Thus, the exponential difference in complexity between \((\mathcal{ACCI}, \mathcal{AQ})\) and \((\mathcal{GF}_2, \mathcal{AQ})\) that we observe for cliquewidth does not exist for treewidth.

The above results concern AQs, and transitioning to (U)CQs brings about some interesting differences. We show that in \((\mathcal{ACCI}, \text{UCQ})\), evaluation of an OMQ \(Q\) is possible in time \(2^{O(|Q| \cdot k^2 \cdot \log(|Q|))} \cdot |D|\) where \(\mathcal{O}\) is the ontology in \(Q\) and \(q\) is the query in \(Q\). This is complemented by the result that already in \((\mathcal{ACCI}, \text{CQ})\), a running time of \(2^{2^{|Q|}} \cdot \text{poly}(|D|)\) cannot be achieved even on databases of cliquewidth 3 unless ETH fails. This should be contrasted with the fact that OMQ evaluation in \((\mathcal{ACCI}, \text{CQ})\) and \((\mathcal{ACCI}, \text{UCQ})\) is \textsc{ExpTime}-complete in combined complexity on unrestricted databases (Lutz 2008) and thus also in these OMQ languages one has to pay for polynomial running time in \(|D|\) by an exponential increase in overall running time. Note however, that the increase is only in \(|q|\), but not in \(|\mathcal{O}|\). This is in contrast to the case of AQs where the size of queries is actually constant, and it is good news as queries tend to be much smaller than ontologies.

Finally we observe that, in \((\mathcal{ACCI}, \mathcal{AQ})\), where \(\mathcal{ACCI}\) is the extension of \(\mathcal{ACCI}\) with functional roles, OMQ evaluation on databases of treewidth 2 is \textsc{coNP}-hard in data complexity when the unique name assumption is not made.

Proof details are delegated to the appendix, see (Lutz, Sabelle, and Schulze 2022).

**Related Work.** Monadic second-order logic (MSO) comes in two versions: MSO\(_1\) admits quantification only over sets of nodes while MSO\(_2\) can additionally quantify over sets of edges. Courcelle’s theorem states that MSO\(_2\) model checking is FPL on graphs of bounded treewidth and (Courcelle,

\(^1\)We explain our use of the \(O\)-notation in the appendix.

<table>
<thead>
<tr>
<th></th>
<th>Bnd. CW</th>
<th>Bnd. TW</th>
<th>Unrestr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\mathcal{ACCI}, \mathcal{AQ}))</td>
<td>\text{snl exp}</td>
<td>\text{snl exp}</td>
<td>\text{ExpTIME}</td>
</tr>
<tr>
<td>((\mathcal{GF}_2, \mathcal{AQ}))</td>
<td>\text{dbl exp}</td>
<td>\text{snl exp}</td>
<td>\text{ExpTIME}</td>
</tr>
<tr>
<td>((\mathcal{ACCI}, \text{UCQ}))</td>
<td>\text{dbl exp}</td>
<td>\text{snl exp}</td>
<td>\text{ExpTIME}</td>
</tr>
<tr>
<td>((\mathcal{ACCI}, \text{UCQ}))</td>
<td>\text{open}</td>
<td>\text{dbl exp}</td>
<td>\text{2ExpTIME}</td>
</tr>
</tbody>
</table>

Table 1: Overview of results

Makowsky, and Rotics (2000) shows the same for MSO\(_1\) and graphs of bounded cliquewidth (when a \(k\)-expression is provided). The problem studied in the current paper can be translated into MSO\(_2\), but this does not yield the tight complexities presented here. It was shown in (Kreutzer and Tzari 2010) that MSO\(_2\) is not \textsc{FPT} for graph classes that have unbounded treewidth and are closed under substructures (under certain assumptions), see also (Ganian et al. 2014); results in this style do not appear to be known for MSO\(_1\) and cliquewidth.

## 2 Preliminaries

**Description Logics.** Let \(N_C, N_R,\) and \(C\) be countably infinite sets of concept names, role names, and constants. A role is a role name \(r\) or an inverse role \(r^{-1}\), where \(r\) is a role name. Set \((r^{-1})^\perp = r.\) \(\mathcal{ACCI}\)-concepts are generated by the rule 

\[ C, D ::= A | ¬ C | C \cap D | C \cup D | \exists r.C | \forall r.C \]

where \(A\) ranges over concept names and \(r\) over roles. An \(\mathcal{ACCI}\)-concept is an \(\mathcal{ACCI}\)-concept that does not use inverse roles. We use \(\top\) to denote \(A \cup \neg A\) for some fixed concept name \(A\) and \(\bot\) for \(\neg \top\). For \(L \in \{\mathcal{ACCI}, \mathcal{ACCI}\}\), an \(L\)-ontology is a finite set of concept inclusions (CIs) \(C \subseteq D\) with \(C\) and \(D\)-concepts. A database is a finite set of facts of the form \(A(c)\) or \(r(c, c')\) where \(A \in N_C \cup \{\top\}, r \in N_R,\) and \(c, c' \in C\). We use \(\text{adm}(D)\) to denote the set of constants used in a database \(D\), also called its active domain. The size of a syntactic object \(X\), denoted \(|X|\), is the number of symbols needed to write \(X\) as a word over a finite alphabet using a suitable encoding.

The semantics is given in terms of interpretations \(\mathcal{I} = (\Delta^X, \mathcal{I})\), we refer to (Baader et al. 2017) for details. An interpretation \(\mathcal{I}\) satisfies a CI \(C \subseteq D\) if \(C^\mathcal{I} \subseteq D^\mathcal{I}\), a fact \(A(c)\) if \(c \in A^\mathcal{I}\), and a fact \(r(c, c')\) if \((c, c') \in r^\mathcal{I}\). We thus make the standard names assumption, that is, we interpret constants as themselves. This implies the unique name assumption. For \(S \subseteq \Delta^X\), we use \(\mathcal{I} \models S\) to denote the restriction of \(\mathcal{I}\) to domain \(S\). An interpretation \(\mathcal{I}\) is a model of an ontology or database if it satisfies all inclusions or facts in it. A database \(D\) is \textsc{satisfiable} w.r.t. an ontology \(\mathcal{O}\) if there is a model \(\mathcal{I}\) of \(\mathcal{O}\) and \(D\).

We also consider the guarded two-variable fragment (GF\(_2\)) of first-order logic. In GF\(_2\), only two fixed variables \(x\) and \(y\) are available and quantification is restricted to the pattern 

\[ \forall \overline{y}(\alpha(\overline{x}, \overline{y}) \rightarrow \varphi(\overline{x}, \overline{y})) \exists \overline{y}(\alpha(\overline{x}, \overline{y}) \land \varphi(\overline{x}, \overline{y})) \]

where \(\varphi(\overline{x}, \overline{y})\) is a GF\(_2\) formula with free variables among \(\overline{x} \cup \overline{y}\) and \(\alpha(\overline{x}, \overline{y})\) is an atomic formula (possibly an equality) called the guard that uses all variables in \(\overline{x} \cup \overline{y}\). Function symbols and constants are not admitted, and neither is equal-
ity in non-guard positions. We only admit relation symbols of
arity one and two, identifying the former with concept
names and the latter with role names. We may thus interpret
$GF_2$ formulas in DL interpretations. A $GF_2$-ontology is a
finite set of $GF_2$-sentences.

Queries. A conjunctive query ($CQ$) is of the form
$q(\bar{x}) = \exists \bar{y} \varphi(\bar{x}, \bar{y})$, where $\bar{x}$ and $\bar{y}$ are
 tuples of variables and $\varphi(\bar{x}, \bar{y})$ is a conjunction of concept
atoms $A(\bar{x})$ and role atoms $r(x, y)$, $A \in NC, r \in NR$, and $x, y$ variables from $\bar{x}\cup\bar{y}$. We call the variables in $\bar{x}$ the answer variables of $q$, and use $\text{var}(q)$ to denote $\bar{x} \cup \bar{y}$. We may write $\alpha \in q$ to indicate that $\alpha$ is an atom in $q$. For $V \subseteq \text{var}(q)$, we use $q|_V$ to denote the restriction of $q$ to the atoms that use only variables in $V$.

A tuple $d \in (\Delta^{|I|})^{|I|}$ is an answer to $q$ on an interpretation $I$ if there is a homomorphism $h$ from $q$ to $I$ with $h(\bar{x}) = d$. More details are in the appendix.

A union of conjunctive queries (UCQ) $q(\bar{x})$ is a disjunction of CQs with the same answer variables $\bar{x}$. A tuple $d \in (\Delta^{|I|})^{|I|}$ is an answer to $q$ on interpretation $I$, written $I |= q(d)$, if $d$ is an answer to some CQ in $q$ on $I$. We use $q(I)$ to denote set of all answers to $q$ on $I$. The arity of $q$ is the length of $\bar{x}$ and $q$ is Boolean if it is of arity zero. An atomic query ($AQ$) is a CQ of the form $A(\bar{x})$ with $A$ a concept name.

Ontology-Mediated Querying. An ontology-mediated query ($OMQ$) is a pair $Q = (O, q)$ with $O$ an ontology and $q$ a query such as a UCQ. While OMQs are often defined to include an additional third component, the data signature, the problem of query evaluation studied in this paper is insensitive to that component, and so we omit it. We write $Q(\bar{x})$ to indicate the answer variables of $q$ are $\bar{x}$. A tuple $\bar{a} \in \text{adom}(D)^{|I|}$ is an answer to $Q(\bar{x})$ on a database $D$, written $D |= Q(\bar{a})$, if $I |= q(\bar{a})$ for all models $I$ of $O$ and $D$. When more convenient, we might alternatively write $D, I |= q(\bar{a})$. We write $Q(D)$ to denote the set of all answers to $Q$ on $D$. We use $(L, Q)$ to denote the OMQ language that contains all OMQs $Q$ in which $O$ is formulated in the DL $L$ and $q$ in the query language $Q$, such as in ($ALCI, UCQ$) and ($GF_2, AQ$).

Let $(L, Q)$ be an OMQ language. We write OMQ evaluation in $(L, Q)$ to denote the problem to decide, given an OMQ $Q(\bar{x}) \in (L, Q)$, a database $D$, and a tuple $\bar{c} \in \text{adom}(D)^{|I|}$, whether $\bar{c} \in Q(D)$. A parameterization is a function $\kappa$ that assigns to every input $Q, D, \bar{c}$ a parameter $\kappa(Q, D, \bar{c}) \in \mathbb{N}$. We will use $\kappa = |Q|$ in lower bounds and $\kappa = |Q| + k$ in upper bounds where $k$ is the cliquewidth of $D$ as defined below. OMQ evaluation is fixed-parameter tractable (FPT) for $\kappa$ if it can be decided by an algorithm that runs in time $f(\kappa(Q, D, \bar{c})) \cdot \text{poly}(n)$ where $f$ is a computable function, here and throughout the paper, poly denotes an unspecified polynomial, and $n$ is the size of the input. Fixed-parameter linearity (FPL) for $\kappa$ is defined analogously, with running time $f(\kappa(Q, D, \bar{c})) \cdot O(n)$.

Treewidth and Cliquewidth. Treewidth describes how similar a graph (in our case: a database) is to a tree, and cliquewidth does the same for cliques (Courcelle and Engelfriet 2012). Due to space restrictions and since this article focuses on cliquewidth, we provide the definition of treewidth only in the appendix. Databases of cliquewidth $k$ can be con-
struced by a $k$-expression as defined below, where $k$ refers to
the number of labels in the expression. For convenience, we use a reservoir of fresh concept names $L_1, L_2, \ldots$ as labels that may occur in databases, but not in ontologies and queries. We say that a constant $c \in \text{adom}(D)$ is labeled $i$ if $L_i(c) \in D$. An expression $s$ defines a database $D_s$ built from four operations, as follows:

$s = \vee(D)$ is the nullary operator for constant introduction
where $i \geq 1$ and $D$ is a database with $|\text{adom}(D)| = 1$ that contains exactly one fact of the form $L_i(c)$, and $D_s = D$.

$s = s_1 \uplus s_2$ is the binary operator for disjoint union
where $s_1$ and $s_2$ are expressions such that $\text{adom}(D_{s_1}) \cap \text{adom}(D_{s_2}) = \emptyset$ and $D_s = D_{s_1} \uplus D_{s_2}$.

$s = \alpha_{i,j}^r(s')$ is the unary operator for role insertion
where $r$ is a role name, $i, j \geq 1$ are distinct, and $s'$ is an expression. It links all constants labeled $i$ to all constants labeled $j$ using role $r$, that is,

$D_s = D_{s'} \cup \{(a, b) \mid L_i(a) \in D_{s'} \land L_j(b) \in D_{s'}\}$.

An expression $s$ is a $k$-expression if all labels in $s$ are from $\{L_1, \ldots, L_k\}$. A database $D$ has cliquewidth $k > 0$ if there is a $k$-expression $s$ with $D_s = D$, but no $k'$-expression $s'$ such that $k' < k$ and $D_s = D$, up to facts of the form $L_i(c)$.

Example 1. Consider the class of databases about schools that use the concept names Pupil, Teacher and School as well as role names teaches, worksAt and isClassmateOf, and where each teacher works at exactly one school and can teach any number of pupils. The set of all pupils is partitioned into groups of pupils that are classmates of one another, and each teacher teaches exactly one such group. A basic database from that class is

$D = \{\text{Pupil}(a_1), \text{Pupil}(a_2), \text{Teacher}(b), \text{School}(c), \text{worksAt}(b, c), \text{teaches}(b, a_1), \text{teaches}(b, a_2), \text{isClassmateOf}(a_1, a_2), \text{isClassmateOf}(a_2, a_1)\}$.

This database has cliquewidth 3, and in fact so does any database from the described class, independently of the number of pupils, teachers, and schools. When constructing $k$-expressions, it suffices to use up to two labels for pupils, one for teachers, and one for schools. With proper relabeling, no more than three labels are needed at the same time.

3 Upper Bounds and Algorithms

We prove that OMQ evaluation on databases of bounded cliquewidth is FPL when the parameter is the size of the OMQ plus the cliquewidth of the database and establish upper bounds on the overall running time.

3.1 $ALCI$ with AQS

Theorem 1. In ($ALCI, AQ$), an OMQ $Q = (O, q)$ can be evaluated on a database $D$ of cliquewidth $k$ in time $2^{O(|O|)} \cdot |D|$.
Theorem 1 implies that OMQ evaluation in \((ALCI, AQ)\) is in FPL, and also that it is in linear time in database size when the cliquewidth of databases is bounded by a constant.

As announced before, we assume here and in all subsequent upper complexity bounds that a \(k\)-expression \(s_0\) for \(D\) is given as part of the input. In fact, the number of subexpressions of \(s_0\) must be \(O(|D|)\), a condition that is satisfied by any reasonable \(k\)-expression for \(D\). These assumptions could clearly be dropped if, given a database \(D\) of cliquewidth \(k\), we could compute a \(k\)-expression that generates \(D\) (or a sufficient approximation thereof) in time \(2^{O(k)} \cdot |D|^3\). It is an open problem whether this is possible, see e.g. (Downey and Fellows 2013).

There is, however, an algorithm that computes, given a database \(D\) of cliquewidth \(k\), in time \(|\text{adom}(D)|^k \cdot k^k\) a \(k\)’-expression with \(k’ \leq 2^{3k-1}\) (Oum 2008). As a consequence, we obtain an analogue of Theorem 1 that does not require a \(k\)-expression for \(D\) to be given and where the time bound is replaced with \(2^{O(k)} \cdot 2^{O(k)} \cdot |D|^3\). While this no longer yields FPL, it still yields FPT.

To prove Theorem 1, it suffices to give an algorithm for database satisfiability w.r.t. an \(ALCI\) ontology that runs within the stated time bounds. To decide whether \(D, O \models A(c)\), we may then simply check whether \(D \cup \{ \overline{A}(c) \}\) is unsatisfiable w.r.t. the ontology \(O \cup \{ \overline{A} = \neg A \}\).

Assume that we are given as input a database \(D_0\), an ontology \(O\), and a \(k\)-expression \(s_0\) that generates \(D_0\). We may assume w.l.o.g. that \(O\) takes the form \(\{ \top \subseteq C_0 \}\) where \(C_0\) is an \(ALCI\)-concept in negation normal form (NNF), that is, negation is only applied to concept names, but not to compound concepts. Every \(ALCI\)-ontology \(O\) can be converted into this form in linear time (Baader et al. 2017).

Our algorithm traverses the \(k\)-expression \(s_0\) bottom-up, computing for each subexpression \(s\) a succinct representation of the models of the database \(D_s\) and ontology \(O\). Before giving details, we introduce some relevant notions. We use \(\text{sub}(O)\) to denote the set of all concepts in \(O\), closed under subconcepts and \(\text{c1}(O)\) for the extension of \(\text{sub}(O)\) with the NNF of all concepts \(\neg C, C \in \text{sub}(O)\). Moreover, \(\text{c1}^+(O)\) denotes the restriction of \(\text{c1}(O)\) to concepts of the form \(\forall r.C \in \text{c1}^+(O)\) is \(\{ C \mid \forall r.C \in \text{c1}(O) \}\). A type for \(O\) is a set \(\in \text{c1}(O)\) such that for some model \(I\) of \(O\) and some \(d \in \Delta^I, t = \{ C \in \text{c1}(O) \mid d \in \Delta^I \}\). We then also denote \(t\) with \(\text{tp}_I(d)\).

An input output assignment (IOA) for \(O\) is a pair \(\gamma = (in, out)\) with \(in : \{1, \ldots, k\} \rightarrow 2^{\text{c1}^+(O)}\) and \(out : \{0, \ldots, k\} \rightarrow 2^{\text{c1}^+(O)}\) total functions. For easier reference, we use \(\gamma^\text{in}\) to denote \(in\) and \(\gamma^\text{out}\) to denote \(out\). Every database \(D\) and model \(I\) of \(D\) and \(O\) give rise to an IOA \(\gamma_{I,D}\) for \(O\) defined by, for \(1 \leq i \leq k\),
\[
\gamma^\text{in}_{I,D}(i) = \text{c1}^+(O) \cap \bigcap_{L_i \in D} \text{tp}_I(c)
\]
\[
\gamma^\text{out}_{I,D}(i) = \text{c1}(O) \cap \bigcup_{L_i \in D} \text{tp}_I(c).
\]

Intuitively, \(\gamma^\text{out}(i)\) lists outputs generated by label \(i\) in the sense that every concept \(\forall r.C \in \gamma^\text{out}(i)\) ‘outputs’ \(C\) to all constants labeled \(j\) when we use the \(\alpha^i_{c,j}\) operation to introduce new role edges. It is then important that \(C \in \gamma^\text{in}(j)\), and in this sense \(\gamma^\text{in}(j)\) lists inputs accepted by label \(j\).

The central idea of our algorithm is to compute, for each subexpression \(s\), the set of IOAs
\[
\Theta(s) = \{ \gamma_{I,D} \mid I\text{ model of } D_s \text{ and } O \}.
\]

It then remains to check whether \(\Theta(s_0)\) is non-empty. The sets \(\Theta(s)\) are computed as follows:
\[
s = \epsilon(D) : \text{Set } \Theta(s) = \{ \gamma_{I,D} \mid I\text{ model of } D \text{ and } O \}.
\]
\[
s = s_1 \oplus s_2 : \text{Set } \Theta(s) = \{ \gamma_{I,D} \mid I\text{ model of } D \text{ and } O \}.
\]
\[
s = \alpha^i_{c,j}(s') : \text{Set } \Theta(s) = \{ \gamma_{I,D} \mid I\text{ model of } D \text{ and } O \}.
\]

We prove in the appendix that the algorithm is correct.

**Lemma 1.** For all subexpressions \(s\) of \(s_0\), \(\Theta(s) = \{ \gamma_{I,D} \mid I\text{ model of } D_s \text{ and } O \} \).

The presented algorithm achieves the intended running time. We do a single bottom-up pass over \(s_0\), considering each subexpression \(s\), of which there are at most \(O(|D|)\) many. The size of each IOA in \(\Theta(s)\) is \(O(|D| \cdot k)\) and there are at most \(2^{O(|D| \cdot k)}\) IOAs. It can be verified that each set \(\Theta(s)\) can be constructed in time \(2^{O(|D| \cdot k)}\). For the case \(s = \epsilon(D)\), this essentially amounts to enumerating all types for \(O\), more details are in the appendix.

### 3.2 \(ALCI\) with UQCs

**Theorem 2.** In \((ALCI, UCQ)\), an OMQ \(Q = (O, q)\) can be evaluated on a database \(D\) of cliquewidth \(k\) in time \(2^{O(|D^{O\land q}(\gamma)|)} \cdot |D|^3\).

The algorithm used to prove Theorem 2 is a generalization of the one used for Theorem 1. It also traverses the \(k\)-expression \(s_0\) bottom-up, computing for each subexpression \(s\), a set of IOAs that additionally are annotated with information about partial homomorphisms from CQs in the UCQ \(q\) into models of \(D_s\) and \(O\). The generalization is not entirely straightforward. First, the homomorphism may map some variables to elements of the model that are ‘outside’ of the database \(D_s\) while IOAs only store information about constants from \(D_s\). And second, we cannot even memorize all homomorphisms that map all variables into \(D_s\) as there may be \(|D_s|^{|\gamma|}\) many of them, too many for FPL. Our solution to the first issue is to modify the ontology and UQC...
so that variables need never be mapped outside of $D$. We resolve the second issue by the observation that it suffices to memorize the labels of the constants in $D$, to which variables are mapped, but not their precise identity. The generalized algorithm establishes Theorem 1 as a special case; we singled out the algorithm in the previous section for didactic reasons.

Assume that we are given as input an OMQ $Q(x) = (O,q) \in (ACL\mathcal{T},\text{UCQ})$, a database $D_0$, and a $k$-expression $s_0$ that generates $D_0$. It is easy to see that we may assume w.l.o.g. that $Q$ is Boolean, details are in the appendix. We may also assume the ontology $O$ to contain a $\mathcal{C}I \subset \bigcup A_T$, with $A_T$ not used anywhere else in $O$, and that every CQ $p$ in $q$ contains the concept atom $A_T(x)$ for every $x \in \var(p)$.

This will prove useful for the constructions below.

An interpretation $\mathcal{I}$ is a tree if the undirected graph $G_{I} = (V,E)$ with $V = D^\Delta$ and $E = \{(x,y) \mid (x,y) \in r^d \text{ for some } r\}$ is a tree and there are no self loops and multi-edges, the latter meaning that $(d,e) \in r^e$ implies $(d,e) \notin r^e$ for all distinct roles $r_1,r_2$. We call a model $\mathcal{I}$ of $D_0$ tree-extended if it satisfies the following conditions:

- $r^I \cap (\text{adom}(D_0) \times \text{adom}(D_0)) = \{(d,e) \mid r(d,e) \in D_0\}$ for all role names $r$ and
- if $\mathcal{I}$ is modified by setting $r^I = r^I \setminus (\text{adom}(D_0) \times \text{adom}(D_0))$ for all role names $r$, then the result is a disjoint union of trees and each of these trees contains exactly one constant from $\text{adom}(D_0)$.

The following is well-known, see for example (Lutz 2008).

**Lemma 2.** If there is a model $\mathcal{I}$ of $D_0$ and $O$ such that $\mathcal{I} \not\models q$, then there is such a model $\mathcal{I}$ that is tree-extended.

Lemma 2 provides the basis for the announced modification of $O$ and the CQs in $q$: we extend $O$ to an ontology $O_q$ and $q$ into a UCQ $\tilde{q}$ such that for every tree-extended model $\mathcal{I}$ of $D_0$ and $O_q$, $\mathcal{I} \models \tilde{q}$ iff $\mathcal{I}|_{\text{adom}(D_0)} \models \tilde{q}$. Then we proceed to work with $O_q$ and $\tilde{q}$ in place of $O$ and $q$, which allows us to concentrate on homomorphisms from CQs in $\tilde{q}$ to models of $D$ and $O_q$, that map all variables to $\text{adom}(D_0)$.

We next present the extension of $O$. A Boolean or unary CQ $p$ being a tree (without self-loops and multi-edges) is defined in the expected way. Note that a unary tree CQ $p$ can be viewed as an AC\mathcal{T}C-concept $C_p$ in an obvious way. For example, if $p(x) = \{A(x), r(y,x), s(y,z), B(z), r(y,z), A(z')\}$, then $C_p = \mathcal{A} \ni \exists r^-.(\exists s,B \ni \exists r^A)$. We use $\texttt{trees}(q)$ to denote the set of all (Boolean) tree CQs that can be obtained from a CQ in $q$ by first dropping atoms and then taking a contraction. When writing $p(x)$ with $p \in \texttt{trees}(q)$ and $x \in \var(p)$, we mean $p$ with $x$ made answer variable. The ontology $O_q$ is obtained from $O$ by adding the following, and then converting to NNF:

- for every $p \in \texttt{trees}(q)$ and $x \in \var(p)$, the CIs $A_p(x) \subseteq \mathcal{C}_p(x)$ and $C_p(x) \subseteq \mathcal{A}_p(x)$ where $A_p(x)$ is a fresh concept name;
- for every $p \in \texttt{trees}(q)$, the CIs $A_p \subseteq \exists r.p_c, C_p \subseteq A_p$, and $\exists r.A_p \subseteq A_p$ for every role $r$ used in $O$, where $A_p$ is a fresh concept name and $r_p$ a fresh role name.

Note that $|O_q| = |O| \cdot 2^(O(|q|)\cdot\log(|q|))$.

Towards defining the UCQ $\tilde{q}$, consider all CQs that can be obtained as follows. Start with a contraction of a CQ from $q$, then choose a set of variables $S \subseteq \var(p)$ such that

$$p^- = p \setminus \{(x,y) \in p \mid x,y \in S\}$$

is a disjoint union of tree CQs each of which contains at most one variable from $S$ and at least one variable that is not from $S$. Include in $\tilde{q}$ all CQs that can be obtained by extending $p|_S$ as follows:

1. for every maximal connected component $p'$ of $p^-$ that contains a (unique) variable $x_0 \in S$, add the atom $A_p(x_0)(x_0)$;
2. for every maximal connected component $p'$ of $p^-$ that contains no variable from $S$, add the atom $A_p'(z)$ with $z$ a fresh variable.

It is easy to verify that in Points 1 and 2 above, the CQ $p'$ is in $\texttt{trees}(q)$ and thus CIs for $p'$ have been introduced in the construction of $O_q$. The number of CQs in $\tilde{q}$ is single exponential in $|q|$ and each CQ is of size at most $|q|$.

**Lemma 3.** For every tree-extended model $\mathcal{I}$ of $D_0$ and $O_q$, $\mathcal{I} \models q \iff 2^{\text{adom}(D_0)} \models \tilde{q}$.

Let $Q$ denote the set of all subexpressions $p$ of CQs in $\tilde{q}$, that is, $p$ can be obtained from a CQ in $\tilde{q}$ by dropping atoms. A decorated IOA for $O_q$ is a triple $\gamma = (\text{in}, \text{out}, S)$ where $(\text{in}, \text{out})$ is an IOA and $S$ is a set of pairs $(p,f)$ with $p \in Q$ and $f : \var(p) \rightarrow \{1,\ldots,k\}$ a total function. We use $\gamma^\text{in}$ to denote in and likewise for $\gamma^\text{out}$ and $\gamma^S$. We argue in the appendix that the number of decorated IOAs is $2^{O(\log(|q|))}$.

Every database $D$ and model $\mathcal{I}$ of $D$ and $O_q$ give rise to a decorated IOA $\gamma_{\mathcal{I},D}$ for $O_q$ where

- $\gamma^\text{in}_{\mathcal{I},D}$ and $\gamma^\text{out}_{\mathcal{I},D}$ are defined as in the previous section and
- $\gamma^S_{\mathcal{I},D}$ contains all pairs $(p,f)$ with $p \in Q$ and $f$ a function from $\var(p)$ to $\{1,\ldots,k\}$ such that there is a homomorphism $h$ from $p$ to $\mathcal{I}$ with $f(x) = i$ iff $L_i(h(x)) \in D$ for all variables $x \in \var(p)$.

For all subexpressions $s$ of $s_0$, our algorithm computes

$$\Theta(s) = \{\gamma_{\mathcal{I},D} \mid \mathcal{I} \text{ is a tree-extended model of } D \text{ and } O_q\}.$$

By Lemma 3 and since every tree-extended model of $D_0$ and $O$ can be extended to a tree-extended model of $D_0$ and $O_q$, this implies that, as desired, $D_0, O \not\models q$ iff there is a decorated IOA $\gamma \in \Theta(s_0)$ such that $\gamma^S$ contains no pair $(p,f)$ with $p$ a CQ in $\tilde{q}$. The sets $\Theta(s)$ are computed as follows:

$$s = \iota(D) : \Theta(s) = \{\gamma_{\mathcal{I},D} \mid \mathcal{I} \text{ is a tree-extended model of } D \text{ and } O_q\}.$$
\( \gamma^S \) to be \( \gamma^S \) extended with all pairs \((p', f)\) such that \( p' \in Q \) and for some \((p, f) \in \tilde{S}, p' \) can be obtained from \( p \) by adding zero or more atoms \( r(x,y) \), subject to the condition that \( f(x) = i \) and \( f(y) = j \).

\[ s = p_{i \rightarrow j}(\gamma'(s')) : \text{for each } \gamma' \in \Theta(s'), \Theta(s) \text{ contains the decorated IOA } \gamma \text{ obtained by defining } \]

\[ \gamma^{in} \text{ and } \gamma^{out} \text{ according to Conditions c) to e) in Section 3.1:} \]

\[ \gamma^S \text{ to contain all pairs } (p, f') \text{ such that for some } (p, f) \in \tilde{S}, f' \text{ can be obtained by starting with } f \text{ and then setting } f'(x) = j \text{ whenever } f(x) = i. \]

Note that \( Q \) contains \( A_T(x) \) for every variable \( x \) in \( q \) and thus we can start with mapping subqueries that contain only a single variable \( x \) (and a dummy atom) in the \( s = \iota(D) \) case. The algorithm achieves the intended goal.

**Lemma 4.** For all subexpressions \( s \) of \( s_0 \), \( \Theta(s) = \{ \gamma_{\mathcal{I}, \mathcal{D}} | \mathcal{I} \text{ is a tree-extended model of } \mathcal{D}_s \text{ and } \mathcal{O}_q \}. \)

The running time is analyzed in the appendix.

### 3.3 GF\(_2\) with AQs

**Theorem 3.** In \((GF_2, AQ), \) an OMQ \( Q = (\mathcal{O}, q) \) can be evaluated on a database \( \mathcal{D} \) of cliquewidth \( k \) in time \( 2^{O(k)} \cdot k^2 \cdot |\mathcal{D}|. \)

As in Section 3.1, we may concentrate on the satisfiability of databases w.r.t. ontologies. To decide whether \( \mathcal{D}, \mathcal{O} \models A(\bar{c}) \), we then simply check whether \( \mathcal{D} \cup \{ \bar{A}(\bar{c}) \} \) is unsatisfiable w.r.t. the ontology \( \mathcal{O} \cup \{ \forall x (\bar{A}(x) \rightarrow \neg A(\bar{x})) \}. \)

Note that this also captures ‘binary AQs’ \( A(\bar{c}) \).

Assume that we are given as input a GF\(_2\)-ontology \( \mathcal{O} \), a database \( \mathcal{D}_0 \), and a \( k \)-expression \( s_0 \) that generates \( \mathcal{D}_0. \) For \( i \in \{1, 2\} \), we use \( \mathcal{C}_i(\mathcal{O}) \) to denote the set of all subformulas of \( \mathcal{O} \) with at most \( i \) free variables, closed under single negation. For \( i = 2 \), we additionally assume that \( \mathcal{C}_1(\mathcal{O}) \) contains all formulas \( r(x,y) \) and \( r(y,x) \) with \( r \) a role name that occurs in \( \mathcal{D}_0. \)

An \( i \)-type for \( \mathcal{O} \) is a set \( t \subseteq \mathcal{C}_1(\mathcal{O}) \) such that for some model \( \mathcal{I} \) of \( \mathcal{O} \) and some \( d \in (\Delta^2)^3 \),

\[ t = \{ \varphi(\bar{x}) \in \mathcal{C}_1(\mathcal{O}) | \mathcal{I} \models \varphi(\bar{c}) \}, \]

\( \bar{c} \) a subtuple of \( d \) of length \( |\bar{c}| = 0, \ldots, i \} \).

We then also denote \( t \) with \( \text{tp}_2(t) \) and use \( \text{TP}_1 \) to denote the set of all \( i \)-types for \( \mathcal{O} \). For \( t_1, t_2 \in \text{TP}_1 \) and \( t \in \text{TP}_2 \), we say that \( t_1 \) and \( t_2 \) are compatible with \( t \) and write \( t_1 \sim t_2 \) if there is a model \( \mathcal{I} \) of \( \mathcal{O} \) and \( d_1, d_2 \in \Delta^2 \) such that \( \text{tp}_2(d_i) = t_i \) for \( i \in \{1, 2\} \) and \( \text{tp}_2(d_1, d_2) = t. \) Note that every \( 1 \)-type contains a 0-type and every 2-type contains a 0-type and ontology \( \mathcal{O} \) that for some model \( \mathcal{I} \):

\[ \forall x \forall y r(x,y) \rightarrow \bigwedge_{1 \leq i \leq n} (A_i(x) \leftrightarrow A_i(y)). \]

The 1-types in \( T_i \) may contain any combination of the formulas \( A_1(x), \ldots, A_n(x) \) and for dealing with the \( \alpha^\mathcal{I}_{i,j} \) operator we clearly have to know all of them. However, there are double exponentially many sets of such combinations, and this results in a double exponential upper bound.

We now give details of how we represent models. A multi-edge is a set of atoms \( r(x,y) \) and \( r(y,x) \). Let \( \mathcal{M} \) denote the set of all multi-edges that only use role names that occur in \( \mathcal{D}_0 \). For a database \( \mathcal{D} \) and distinct \( a, b \in \text{edom}(\mathcal{D}) \), we use \( \mathcal{M}_D(a, b) \) to denote the multi-edge that consists of all atoms \( r(x,y) \) such that \( \mathcal{D} \) contains the fact \( r(a,b) \) and \( r(y,x) \) such that \( \mathcal{D} \) contains the fact \( r(b,a) \).

A model abstraction is a pair \( \gamma = (T, E) \) where:

- \( T : \{1, \ldots, k\} \rightarrow \text{TOP} \)
- \( E : \{1, \ldots, k\} \rightarrow 2^\text{TOP} \times \text{ED} \times \text{ED} \)

such that all 1-types in the ranges of \( T \) and \( E \) contain exactly the same subformulas of \( \mathcal{O} \) that are sentences, that is, they contain the same 0-type. We use \( \gamma^0 \) to denote this 0-type, \( \gamma^T \) to denote \( T \), and likewise for \( \gamma^E \). We use multi-edges in model abstractions rather than 2-types as the former allow us to add edges later when the operator \( \alpha^\mathcal{I}_{i,j} \) is applied whereas the latter fix all edges from the beginning.

Every database \( \mathcal{D} \) and model \( \mathcal{I} \) of \( \mathcal{D} \) give rise to a model abstraction \( \gamma_{\mathcal{I}, \mathcal{D}} \) defined by setting, for \( 1 \leq i, j \leq k \),

\[ \gamma_{\mathcal{I}, \mathcal{D}}^T(i) = \{ \text{tp}_2(c) | L_i(c) \in \mathcal{D} \} \]

\[ \gamma_{\mathcal{I}, \mathcal{D}}^E(i, j) = \{ \text{tp}_2(c), M_D(c, c'), \text{tp}_2(c') | L_i(c) \in \mathcal{D} \text{ and } L_j(c') \in \mathcal{D} \}. \]

Our algorithm computes, for each subexpression \( s \) of \( s_0 \),

\[ \Theta(s) = \{ \gamma_{\mathcal{I}, \mathcal{D}} | \mathcal{I} \text{ model of } \mathcal{D}_s \text{ and } \mathcal{O} \}. \]

It then remains to check whether \( \Theta(s_0) \) is non-empty. The sets \( \Theta(s) \) are computed as follows:

**s = s_1 \oplus s_2**: \( \Theta(s) \) contains a model abstraction \( \gamma = (T, F) \) for each pair of model abstractions \( (\gamma_1, \gamma_2) \in \Theta(s_1) \times \Theta(s_2) \) with \( \gamma^0_1 = \gamma^0_2 \) defined by setting, for \( 1 \leq i, j \leq k \),

\[ \gamma^T(i) = \gamma^T_1(i) \cup \gamma^T_2(i) \]

\[ \gamma^E(i, j) = \gamma^E_1(i, j) \cup \gamma^E_2(i, j) \cup \]

\[ \{(t_1, \emptyset, t_2) | t_1 \in \gamma^T_1(i) \text{ and } t_2 \in \gamma^T_2(j)\} \cup \]

\[ \{(t_1, \emptyset, t_2) | t_1 \in \gamma^T_2(i) \text{ and } t_2 \in \gamma^T_1(j)\}. \]
If the ETH is true, then there is no algorithm for the number of clauses in the input formula. For both points of view, that produces an ontology of size \(O(\log m)\) where \(m\) is the number of clauses in the input formula. For both points of view, that produces an ontology of size \(O(\log m)\) where \(m\) is the number of clauses in the input formula.

The model abstraction \(\gamma^T\) defined like \(\hat{\gamma}\) except that 
\[
\gamma^T(i) = \emptyset \quad \gamma^E(i, \ell) = \emptyset \\
\gamma^E(j, \ell) = \gamma^E(j, \ell) \cup \gamma^E(i, \ell) \\
\gamma^E(\ell, j) = \gamma^E(\ell, j) \cup \gamma^E(\ell, i),
\]
The algorithm achieves the intended goal and runs within the stated time bounds.

**Lemma 5.** For all subexpressions \(s\) of \(s_0\), \(\Theta(s) = \{\gamma_I, \mathcal{D}, |T\} model of \mathcal{D}_s \) and \(\mathcal{O}\).

### 4 Lower Bounds

We prove that for \((\mathcal{G}_2, \mathcal{Q})\) and \(\mathcal{L}\), the double exponential running time in \(|Q|\) cannot be improved unless the exponential time hypothesis (ETH) is false. The ETH states that if \(\delta\) is the infimum of real numbers such that \(3\text{SAT}\) can be solved in time \(O(2^{\delta n})\), where \(n\) is the number of variables in the input formula, then \(\delta > 0\). The following are the two main results of this section.

**Theorem 4.** If the ETH is true, then there is no algorithm with running time \(2^{2^{|Q|}} \cdot \text{poly}(|\mathcal{D}|)\) for OMQ evaluation in 1. \((\mathcal{G}_2, \mathcal{Q})\) on databases of cliquewidth \(2\); 2. \((\mathcal{L}, \mathcal{C})\) on databases of cliquewidth \(3\).

Note that OMQ evaluation in \((\mathcal{G}_2, \mathcal{Q})\) and \((\mathcal{L}, \mathcal{C})\) are only \textsc{ExpTime}-complete in combined complexity on unrestricted databases. The former is folklore and the latter is proved in (Lutz 2008). Thus, Theorem 4 implies that, in these two OMQ languages, one has to pay for the running time to be polynomial in \(|\mathcal{D}|\) on databases of bounded cliquewidth by an exponential increase in the running time in \(|Q|\).

We now prove Theorem 4. A major contribution of the original paper bringing forward the ETH is the sparsification lemma, which implies the following useful consequence of the ETH (Impagliazzo, Paturi, and Zane 2001):

(*) If the ETH is true, then \(3\text{SAT}\) cannot be solved in time \(2^{2^{|m|}}\), with \(m\) the number of clauses of the input formula.

Both results in Theorem 4 are proved by contradiction against (*), i.e. we show that an algorithm for these OMQ evaluation problems with running time \(2^{2^{|m|}} \cdot \text{poly}(|\mathcal{D}|)\) can be used to design an algorithm for \(3\text{SAT}\) with running time \(2^{2^{|m|}}\), contradicting (*). To achieve this, it suffices to give a polynomial time reduction from \(3\text{SAT}\) to OMQ evaluation that produces an ontology of size \(O(\log m)\) where \(m\) is the number of clauses in the input formula. For both points of view, that produces an ontology of size \(O(\log m)\) where \(m\) is the number of clauses in the input formula.
evaluation in \((\text{GF}_2, \text{AQ})\) on databases of cliquewidth 2, where the constructed ontology is of size \(O(\log(k))\) with \(k\) the number of distinct colors of the input graph.

We are now ready to prove Theorem 4.

**Proof.** Assume for a contradiction that there is an algorithm with running time \(2^{2^{\Theta(n^2)}} \cdot \text{poly}(|D|)\) for OMQ evaluation in \((\text{GF}_2, \text{AQ})\) on databases of cliquewidth at most 2. Then the following is an algorithm for 3SAT. Given a formula in 3CNF with \(m\) clauses, apply the reduction from Theorem 5 followed by the one from Lemma 6. This yields an OMQ \(Q \in (\text{GF}_2, \text{AQ})\) where the ontology \(O\) is of size \(O(\log(m))\), a database \(D\) of cliquewidth 2 and an answer candidate \(a \in \text{adom}(D)\). Use the algorithm for OMQ evaluation in \((\text{GF}_2, \text{AQ})\) whose existence we assumed on input \(Q, D\) and \(a\). The running time of the first step is polynomial. The running time of the second step is \(2^{2^{\Theta(n^2)}} \cdot \text{poly}(|D|)\), which is \(2^{\Theta(n^2)}\), contradicting the ETH. 

Interestingly, the reduction does not rely on the additional expressive power of GF2 compared to \(\text{ALC}\), but rather on the ability of GF2 to express certain statements more succinctly. To demonstrate this, we introduce the extension \(\text{ALC}^m\) of \(\text{ALC}\) with the following constructor:

If \(A_1, \ldots, A_n\) are concept names, then \(\exists r.A_1, \ldots, A_n\] is a concept. The semantics is defined as follows:

\[\exists r.\exists \vec{v}.[A_1, \ldots, A_n]_{\vec{v}} = \{d \in \Delta^\vec{v} : \exists e \in \Delta^\vec{v} : (d, e) \in r^\vec{v} \land d \in A_i^\vec{v} \iff e \in A_i^\vec{v} \text{ for all } i \in \{1, \ldots, n\}\}.

It is easy to see that the expressive power of \(\text{ALC}\) and of \(\text{ALC}^m\) are identical, as the additional constructor \(\exists r.\exists \vec{v}.[A_1, \ldots, A_n]_{\vec{v}}\) can be expressed as:

\[
\bigcup_{c_1 \in \{A_1, \neg A_1\}} \bigcup_{c_n \in \{A_n, \neg A_n\}} (C_1 \equiv \cdots \equiv C_n \equiv r.(C_1 \equiv \cdots \equiv C_n)).
\]

Note that the above concept is exponentially larger than the original one. In GF2, in contrast, the new constructor can easily be expressed using only a linear size formula.

In the reduction used to prove Lemma 6, we may replace the GF2 ontology by the following \(\text{ALC}^m\)-ontology:

\[O = \{\exists s.\exists \vec{v}.[A_1, \ldots, A_n]_{\vec{v}} \subseteq D\} \cup \{B \cup \exists r.\exists \vec{v}.[A_1, \ldots, A_n]_{\vec{v}} \subseteq D\} \cup \{\exists \vec{d} D \cup D \subseteq \forall r.D, \exists s.\exists \vec{d} D \subseteq D, D \subseteq \forall s.D\} \cup \{\top \subseteq A^0_1 \cup A^1_1, A^0_1 \cap A^1_1 \subseteq \bot \mid i \in \{1, \ldots, \ell\}\}.\]

Thus, Point 1 of Theorem 4 still holds if \((\text{GF}_2, \text{AQ})\) is replaced with \((\text{ALC}^m, \text{AQ})\).

To prove Point 2 of Theorem 4, we use a reduction that is similar to the one presented before Lemma 6. This time we are only allowed to use an \(\text{ALC}\) ontology, but also CQs in place of AQs, which again allows us to express succinctly that adjacent nodes have to be colored differently.

**Lemma 7.** There is a polynomial time reduction from List COLORING on connected graphs of cliquewidth 2 to OMQ evaluation in \((\text{ALC}, \text{CQ})\) on databases of cliquewidth 3, where the constructed OMQ is of size \(O(\log(k))\), \(k\) the number of distinct colors of the input graph.

**Proof.** Let \(G = (V, E)\) be an undirected graph of cliquewidth 2 and \(C_v \subseteq \mathbb{N}\) a list of possible colors for every \(v \in V\). We construct a database \(D\) of cliquewidth 3 and a Boolean OMQ \(Q = (O, q)\) from \((\text{ALC}, \text{CQ})\) such that \(G\) and \(C_v\) are yes-instance for List COLORING if and only if \(D \not\models Q\). In contrast to what was described above, we do not carry out a reduction from List COLORING to PRE-COLORING EXTENSION as a separate first step, but build it directly into the main reduction. Let \(k\) be the number of distinct colors that occur, w.l.o.g. let \(\bigcup_{v \in V} C_v = \{1, \ldots, k\}\).

Define \(E = [\log(k)]\) and \(L = \{1, \ldots, 2^E\}\)

We use concept names \(A_i^j\), \(i \in \{1, \ldots, \ell\}\) and \(j \in \{0, 1\}\), another concept name \(B\), one role name \(r\), and constants \(a_v, b_v, a_v^0\), and \(b_v^0\) for every \(v \in V\) and \(c \in L \setminus C_v\).

The database is defined as

\[
\begin{align*}
\{ & r(a_v, a_u), r(a_u, a_v) \mid u, v \in E \}\cup \\
& \{r(a_v, a_v^0), r(a_v^0, a_v) \mid v \in V, c \in L \setminus C_v \}\cup \\
& \{A_i^j(a_v^0) \mid v \in V, c \in L \setminus C_v \text{ and the } \i-th \text{ bit of } c \text{ is } j \}\cup \\
& \{r(a_v, b_v), r(b_v, a_v), r(b_v, b_v) \mid v \in V \}\cup \\
& \{r(a_v^0, b_v^0), r(b_v^0, a_v^0), r(b_v^0, b_v^0) \mid v \in V, c \in L \setminus C_v \}\cup \\
& \{A_i(a_v), A_i(a_v^0) \mid v \in E, c \in L \setminus C_v \}\cup \\
& \{B(b_v), B(b_v^0) \mid v \in V, c \in L \setminus C_v \}.
\end{align*}
\]

Here, every constant \(a_v\) represents a vertex of \(G\), and every constant \(a_v^0\) is a constant adjacent to \(a_v\) that is precolored with the color \(c\). Furthermore, for every constant \(a_v^0\) (resp. \(a_v^0\)) there is a constant \(b_v\) (resp. \(b_v^0\)) such that \(b_v\) (resp. \(b_v^0\)) is adjacent only to \(a_v\) (resp. \(a_v^0\)). We refer to the \(b_v\) and \(b_v^0\) as the copies of \(a_v\) and \(a_v^0\). Every copy has an \(r\)-selfloop and a unary marker \(B\), and every non-copy has a unary marker \(A\). Figure 1 depicts a part of the database. We argue in the appendix that \(D\) has cliquewidth 3.

Next, we construct the ontology, which has two purposes. First, we assign a unique color to every constant of the database. For every \(i \in \{1, \ldots, \ell\}\), we include the CIs \(\top \subseteq A^0_i \cup A^1_i\) and \(A^0_i \cap A^1_i \subseteq \bot\). The second purpose is to assure that every color \(b_v\) (resp. \(b_v^0\)) is colored using the anti-color of the color of \(a_v\) (resp. \(a_v^0\)). Here, the *anti-color*
of a color $c$ is the color obtained from $c$ by flipping every bit, e.g. the anti-color of 10101 is 01100. Since the copies $b_v$ and $b'_v$ carry the concept name $B$, this can be achieved using the following CIs for every $i \in \{1, \ldots, \ell\}$ and $j \in \{0, 1\}$:

$$A \cap B \subseteq \bot \quad B \cap A_i^j \subseteq \forall r. (B \cup A_i^{1-j}).$$

Now we construct the (Boolean) CQ $q$, whose purpose it is to detect a defect in the coloring. It is a union of $\ell$ gadgets, with gadget number $i$ checking that the colors of two adjacent vertices agree on bit number $i$. We use two variables $x$ and $y$, as well as variables $z_1^i, z_2^i, \ldots, z_\ell^i$ for every $i \in \{1, \ldots, \ell\}$. The variables $x$ and $y$ are shared between all the gadgets, whereas the variables $z_1^i, z_2^i, \ldots, z_\ell^i$ belong to gadget number $i$. The $i$-th gadget is depicted in Figure 2. It is clear that the OMQ $(\mathcal{O}, q)$ is of size $O(\log k)$.

Claim. $G$ and $C_v$ are yes-instance for LIST COLORING if and only if $D \not\models Q$.

We only show the ‘if’ direction here as it demonstrates how the constructed CQ $q$ can identify defects in the coloring. Let $I$ be a model of $\mathcal{O}$ and $D$ such that $I \not\models q$. We show that $G$ and the $C_v$ are a yes-instance of LIST COLORING. Since $I$ is a model of $\mathcal{O}$, every $a_v$ is assigned a unique color $c_v$, encoded using the $A_i^j$. We define $f(v) = c_v$ for every $v \in V$. If any two adjacent constants $a_u$ and $a_u$ were colored with the same color $c = j_1j_2\ldots j_\ell \in \{0, 1\}^\ell$, then we can construct a homomorphism $h$ from $q$ to $I$ by $h(x) = b_v$, $h(y) = b_u$ and, if $j_i = 0$, then

$$h(z_1^i) = a_v, h(z_2^i) = a_u, h(z_3^i) = b_u$$

and if $j_i = 1$, then

$$h(z_1^i) = b_v, h(z_2^i) = a_v, h(z_3^i) = a_u$$

Figure 1 depicts the case where $c = 101$. But we assumed that $I \not\models q$, so every two adjacent $a_v$ and $a_u$ have different colors. Thus, $f$ witnesses that $G$ and the $C_v$ are a yes-instance for LIST COLORING.

With Lemma 7 at hand, the proof for Point 2 of Theorem 4 continues exactly as the proof of Point 1.

5 Bounded Treewidth

We prove two results that concern OMQ evaluation on databases of bounded treewidth, mainly to contrast with the results obtained for databases of bounded cliquewidth.

Theorem 6. In $(GF_2, \mathcal{O})$, an OMQ $Q = (\mathcal{O}, q)$ can be evaluated on a database $D$ of treewidth $k$ in time $2^{O(\log |\mathcal{O}| \cdot \log(k^3))} \cdot |D|$.

Note that we obtain single exponential running time while Theorem 4 states that when treewidth is replaced by cliquewidth, we cannot be better than double exponential.

Theorem 7. In $(GF_2, UCQ)$, evaluating an OMQ $Q = (\mathcal{O}, q)$ on a database $D$ of treewidth $k$ is possible in time $2^{O(\log |\mathcal{O}| \cdot \log(|\mathcal{O}|))} \cdot |D|$.

Theorem 7 is relevant because we leave open the corresponding case for cliquewidth, that is, we do not even know whether OMQ evaluation in $(GF_2, UCQ)$ on databases of bounded cliquewidth is in PTIME in data complexity. We remark that the overall running time in Theorem 7 cannot be improved to single exponential because OMQ evaluation in $(\mathcal{ALC}, \mathcal{CQ})$ is $2\text{ExpTime}$-complete in combined complexity on databases of the form $(\mathcal{A}(c))$ (Lutz 2007).

We give a single algorithm that yields both Theorem 6 and 7. Assume that we are given as input an OMQ $Q = (\mathcal{O}, q)$ in $(GF_2, UCQ)$, and a database $D$ of treewidth $k$. As in the proof of Theorem 2, we may assume w.l.o.g. that $Q$ is Boolean. In contrast to the case of cliquewidth, we do not need to assume that a tree decomposition is given as part of the input. In fact, a tree decomposition $T = (V, E, (B_v)_{v \in V})$ of $D_0$ of width at most $2k + 1$ can be computed in time $2^{O(k)} \cdot |\text{adom}(D_0)|$ (Korhonen 2021).

Our algorithm traverses the tree decomposition $T$ bottom-up. For every node $v$, we compute a representation of the models of $D_v$ and $\mathcal{O}$, where $D_v$ is the restriction of $D_0$ to the constants that occur in the bags of the subtree of $T$ rooted at $v$. Along with the representations of models, we keep track of partial homomorphisms from CQs in $q$ to these models (we actually rewrite the query beforehand, in the style of Section 3.2). Details are in the appendix.

6 Conclusion

We leave open the interesting question whether OMQ evaluation in $(GF_2, (U)CQ)$ on databases of bounded cliquewidth is in PTIME in data complexity. While all other OMQ languages studied in this paper are easily translated into MSO$_1$, this is not the case for $(GF_2, (U)CQ)$. In fact, it is straightforward to express the problem MONOCHROMATIC TRIANGLE$^3$ in $(GF_2, \mathcal{CQ})$ and to our knowledge it is open whether this problem can be expressed in MSO$_1$.

It would also be interesting to consider natural extensions of $\mathcal{ALC}$. We believe that adding role inclusions has no impact on the obtained results. It is less clear what happens for transitive roles, number restrictions, and nominals. In the appendix, we make the following observation for the extension $\mathcal{ALC}F$ of $\mathcal{ACC}$ with (globally) functional roles.

Theorem 8. OMQ evaluation in $(\mathcal{ALC}F, \mathcal{CQ})$ on databases of treewidth $2$ is $\text{ConP}$-hard in data complexity if the unique name assumption is not made.

The proof relies on an observation from (Figueira 2016). We conjecture that when the unique name assumption is made, fixed-parameter linearity is regained and can be proved using the methods in this paper.

It would also be very interesting to pursue the idea of decomposing databases into a (hopefully small) part of high cliquewidth and parts of low cliquewidth, with the aim of achieving efficiency in practical applications.

Acknowledgements

Supported by the DFG CRC 1320 EASE - Everyday Activity Science and Engineering.

$^3$Can the edges of a given graph be colored with two colors without generating a monochromatic triangle?
References


