Guarded-Based Disjunctive Tuple-Generating Dependencies

Rule-based languages lie at the core of several areas of central importance to databases and artificial intelligence, such as data exchange, deductive databases, and knowledge representation and reasoning, to name a few. An algorithmic task that involves rules and is crucial for the above applications is query answering. In this work, we perform an in-depth complexity analysis of query answering under the main guarded-based classes of disjunctive tuple-generating dependencies (TGDs), focussing on (unions of) conjunctive queries ((U)CQs). We show that the problem under investigation is very hard, namely $2\text{ExpTime}$-complete, even for fixed sets of dependencies of a very restricted form. This is a surprising lower bound that demonstrates the enormous impact of disjunction on query answering under guarded-based TGDs, and also reveals the source of complexity for expressive computational logics such as the guarded fragment of first-order logic. We then proceed to investigate whether prominent subclasses of (U)CQs, i.e., queries of bounded treewidth and hypertree-width, and acyclic queries, have a positive impact on the complexity of the problem under consideration. Interestingly, we show that queries of bounded treewidth and of bounded hypertree-width do not reduce the complexity of our problem, even if we focus on predicates of bounded arity, or on fixed sets of disjunctive TGDs. Regarding acyclic queries, although the problem remains $2\text{ExpTime}$-complete in general, in some relevant settings the complexity reduces to $\text{ExpTime}$-complete; in fact, this requires to bound the arity of the predicates, and for some expressive guarded-based formalisms, to fix the set of dependencies. Finally, with the aim of identifying tractable cases, we focus our attention on atomic queries. We show that atomic queries do not make the query answering problem easier under classes of guarded-based disjunctive TGDs that allow more than one atom to occur in the body of the dependencies. However, the complexity significantly decreases in the case of dependencies that can have only one atom in the body, and in fact we obtain the first (and, to the best of our knowledge, the only known) tractability results for query answering under disjunctive TGDs. In particular, we obtain a $\text{PTime}$-completeness if we focus on predicates of bounded arity, and $\text{AC}^0$-membership when the set of dependencies and the query are fixed. It is worth noting that our techniques and results on query answering under guarded disjunctive TGDs can be used as a generic tool for establishing complexity results for query answering under central description logics.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems — query processing, relational databases; I.2.3 [Artificial Intelligence]: Deduction and Theorem Proving — logic programming, resolution

General Terms: Algorithms, Theory, Languages

Additional Key Words and Phrases: Query answering, conjunctive queries, tuple-generating dependencies, existential rules, disjunction, guardedness, computational complexity

1. INTRODUCTION

Rule-based languages lie at the core of several areas of central importance to databases and artificial intelligence, such as data exchange, deductive databases, and knowledge representation and reasoning, to name a few. An algorithmic task that involves rules and is crucial for the above applications is query answering. The central objective of the current work is to better understand the computational complexity of query answering under a broad class of disjunctive rules.

1.1. Tuple-Generating Dependencies

The well-known tuple-generating dependencies (TGDs) [Beeri and Vardi 1984], which are also known as existential rules [Baget et al. 2011a] and Datalog± rules [Cali et al. 2010], form a prominent rule-based formalism. TGDs are implications of the form $\forall X (\varphi(X) \rightarrow \exists Y (\psi(X,Y)))$, where $\varphi$ and $\psi$ are conjunctions of atoms, and they essentially say that some tuples in a relational instance imply the presence of some other tuples in the instance (hence the term “tuple-generating”). For example, the TGD

$$\forall X (\text{employee}(X) \rightarrow \exists Y (\text{hasManager}(X,Y) \land \text{employee}(Y)))$$
expresses the fact that “each employee has a manager who is also an employee.” TGDs were intended for providing a unifying framework for the large range of relational dependencies introduced and studied in the 1970s and the 1980s. Many of those dependencies, such as inclusion, multivalued and join dependencies, are indeed expressible via TGDs [Abiteboul et al. 1995].

By the late 1980s, a sense of fatigue had settled in concerning the study of database dependencies. In his invited paper at PODS 1987 [Ullman 1987], Jeffrey D. Ullman listed database dependency theory in a section titled “Last Gasp of the Dying Swans”, and stated the following: “Perhaps it is time for work in these areas to be driven more by potential applications than by the need to investigate yet another class of dependencies.” Interestingly, during the last decade, TGDs have found many uses and applications in different areas of database and AI research. For example, they have been used in formalizing and investigating inter-operability tasks such as data exchange [Fagin et al. 2005]. TGDs have been also used in metadata management tasks, and in particular for formalizing operations on schema mappings [Fagin 2007]. Finally, they have been used for knowledge representation and reasoning purposes, and in fact as an alternative way for modeling ontologies [Calì et al. 2010; Baget et al. 2011a].

1.2. Query Answering

Query answering under TGDs is one of the main algorithmic tasks that involves TGDs, and is of special interest for database and AI applications, such as the ones mentioned above. The importance of query answering under constraints has been recognized both by the database and AI communities, and this is documented by a recent Dagstuhl seminar dedicated on querying and reasoning under expressive constraints1.

A prominent class of queries, which is of special interest for the current work, are the well-known conjunctive queries (CQs). CQs are assertions of the form $\exists X (\varphi(X, Y))$, where $\varphi$ is a conjunction of atoms — if $Y = \emptyset$, i.e., there are no free variables, then the above CQ is called Boolean; for example, the query $\exists X (\text{hasManager}(X, \text{Bob}))$, where Bob is a constant, asks whether Bob is a manager. CQs correspond to the select-project-join fragment of relational algebra, and form one of the most natural and commonly used languages for querying relational databases [Abiteboul et al. 1995].

Given a database $D$, a set $\Sigma$ of TGDs, and a Boolean conjunctive query $q = \exists X (\varphi(X))$, the problem of query answering is defined as follows: decide whether each model of the logical theory $(D \land \Sigma)$, i.e., a relational instance that contains $D$ and satisfies $\Sigma$, is also a model of $q$, written as $D \cup \Sigma \models q$. In case $q = \exists X (\varphi(X, Y))$ with free variables $Y$, the query answering problem consists of deciding whether a $|Y|$-tuple of constants $t$ is an answer to $q_t$ over $(D \land \Sigma)$, or, equivalently, whether $D \cup \Sigma \models q_t$, where $q_t = \exists X (\varphi(X, t))$. Unfortunately, CQ answering under TGDs is an undecidable problem [Beeri and Vardi 1981], even in the case of fixed sets of TGDs [Cali et al. 2013], and singleton sets of TGDs [Baget et al. 2011a]. There has been a recent and increasing focus on the development of (semantic and syntactic) conditions that can be applied on the TGDs in order to guarantee the decidability of CQ answering; see, e.g., [Fagin et al. 2005; Krötzsch and Rudolph 2011; Calì et al. 2012b; Leone et al. 2012; Thomazo et al. 2012].

1.3. What about Disjunctive Rules?

Although several expressive TGD-based formalisms, which guarantee the decidability of CQ answering, have been proposed and investigated the last years, none of them is powerful enough for nondeterministic guessing, or, simply, for expressing disjunctive TGDs of the form $\forall X (\varphi(X) \rightarrow \bigvee_{i=1}^n \exists Y_i (\psi_i(X, Y_i)))$, introduced in [Deutsch and Tannen 2003]. (Notice that by allowing disjunction in the left-hand side of a TGD we do

1http://www.dagstuhl.de/14331
not add expressive power since such a rule can be transformed into an equivalent set of TGDs, just by introducing a TGD for each disjunct.) The usefulness of disjunctive TGDs for database and AI applications is generally acknowledged. In what follows, we focus on three such applications, namely metadata management, database querying, and knowledge representation and reasoning, and we briefly discuss how disjunctive TGDs can be used:

(1) **Quasi-Inverses of Schema Mappings.** Schema mappings are high-level specifications that describe the relationship between two database schemas [Fagin et al. 2005], and form the vital building block of crucial data interoperability tasks such as data exchange. An important metadata management task is schema mapping in- version. This is done by applying an inverse operator that, intuitively speaking, transforms a schema mapping \( M \) into a schema mapping \( M' \) such that, if after applying \( M \) we then apply \( M' \), the resulting effect of \( M' \) is to “cancel” the effect of \( M \), i.e., the composition of \( M \) with \( M' \) gives the identity schema mapping [Fagin 2007]. The inverse operator, as described above, turned out to be rather restrictive, since it is rare that a schema mapping has an inverse, and a more refined operator, called quasi-inverse, was proposed and investigated in [Fagin et al. 2008]. One of the main results of [Fagin et al. 2008] was to characterize the language needed to express quasi-inverses of schema mappings; here it comes the significance of disjunctive TGDs. If a schema mapping specified by source-to-target TGDs is quasi-invertible, then it has a quasi-inverse specified by a set of target-to-source disjunctive TGDs; the following example has been taken from [Fagin et al. 2008].

**Example 1.1 (Quasi-Inverse).** Let \( M \) be the schema mapping specified by

\[
\begin{align*}
\forall X \forall Y (s(X,Y) & \rightarrow p(X,Y)) \\
\forall X \forall Y (t(X,Y) & \rightarrow p(X,X)).
\end{align*}
\]

There is only one possible generator of \( p(X,Y) \) if \( X \neq Y \), that is, \( s(X,Y) \), and this is captured by the TGD

\[
\forall X \forall Y (p(X,Y) \land \text{constant}(X) \land \text{constant}(Y) \land \text{neq}(X,Y) \rightarrow s(X,Y)),
\]

where \( \text{constant}(c) \) states that \( c \) is a constant, and \( \text{neq}(c,d) \) states that \( c \neq d \). Furthermore, there are two possible generators of \( p(X,X) \), namely \( s(X,X) \) and \( t(X,Y) \), and this is reflected by the disjunctive TGD

\[
\forall X (p(X,X) \rightarrow s(X,X) \lor \exists Y (t(X,Y))).
\]

In fact, the algorithm for producing quasi-inverses considers all such generators. ■

Apart from computing the quasi-inverse \( M' \), the task of posing a CQ over \( M' \) is also highly relevant for data exchange purposes. In this case, we are actually facing the problem of CQ answering under disjunctive TGDs, which is defined in the same way as the problem of CQ answering under TGDs.

(2) **Database Queries.** There are practical queries that can be easily expressed using disjunctive TGDs. Such a query is whether a graph is 3-colorable. This problem can be naturally encoded as a CQ answering problem under disjunctive TGDs (even without existentially quantified variables).

**Example 1.2 (3-Colorability).** Consider a graph \( G = (V,E) \), and let \( D_G \) be the database that stores \( G \) in the natural way, i.e., we have an atom \( \text{vertex}(v) \) for each...
\( v \in V \), and an atom \( edge(v, u) \) for each edge \((v, u) \in E\). We define \( \Sigma \) as follows:

\[
\forall X (\text{vertex}(X) \rightarrow \text{red}(X) \lor \text{green}(X) \lor \text{blue}(X))
\]

\[
\forall X \forall Y (\text{edge}(X, Y) \land \text{red}(X) \land \text{red}(Y) \rightarrow \text{notLegal})
\]

\[
\forall X \forall Y (\text{edge}(X, Y) \land \text{green}(X) \land \text{green}(Y) \rightarrow \text{notLegal})
\]

\[
\forall X \forall Y (\text{edge}(X, Y) \land \text{blue}(X) \land \text{blue}(Y) \rightarrow \text{notLegal}).
\]

Roughly speaking, the first rule assigns to each vertex of \( G \) one of the three colors. If the assignment is not a legal coloring, then the atom \text{notLegal} is inferred by one of the subsequent rules. It is easy to see that \( G \) is 3-colorable iff \( D \cup \Sigma \not\models \text{notLegal} \).

The set of disjunctive TGDs in the above example is actually a disjunctive Datalog program. Datalog is a famous database query language that uses disjunction- and function-free logic programs; see, e.g., [Ceri et al. 1990; Abiteboul et al. 1995]. Disjunctive Datalog, that is, Datalog extended with disjunction, has been thoroughly investigated in [Eiter et al. 1997], where it was convincingly argued that disjunction is very useful, and actually necessary, for expressing some practical problems, such as the one in Example 1.2. Disjunctive TGDs extend disjunctive Datalog with existential quantification in the right-hand side of the rules, in the same way as TGDs extend (plain) Datalog with existential quantification.

(3) Knowledge Representation and Reasoning. Description Logics (DLs) are languages for knowledge representation and reasoning, which are based on concepts (classes of objects) and roles (binary relations on objects). DLs are amongst the most popular knowledge representation formalisms, and they have been employed in several application areas, such as health and life sciences, natural language processing, and data integration, to name a few; for more applications and details see [Baader et al. 2003]. They also play a crucial role in the Semantic Web, where they provide the logical underpinning for the OWL languages, the standard way for modeling Semantic Web ontologies [Grau et al. 2008]. It is interesting to observe that several key DLs are captured by disjunctive TGDs.

**Example 1.3 (Description Logics).** Consider the following axioms expressed in \( ALC \), one of the central DLs, introduced in [Schmidt-Schauß and Smolka 1991], which is at the basis of many other expressive DLs:

\[
\text{parentOf} \sqsubseteq \exists \text{isparent} \sqsubseteq \exists \text{grandparentOf} \sqsubseteq \text{human}
\]

\[
\text{human} \sqsubseteq \text{male} \cup \text{female}.
\]

The former states that “each parent of a father is the grandparent of a human”, while the latter states that “each human is a male or a female”. It is easy to verify that the same can be expressed using the following disjunctive TGDs:

\[
\forall X \forall Y (\text{parentOf}(X, Y) \land \text{isparent}(Y) \rightarrow \exists Z (\text{grandparentOf}(X, Z) \land \text{human}(Z)))
\]

\[
\forall X (\text{human}(X) \rightarrow \text{male}(X) \lor \text{female}(X)).
\]

Notice that for expressing DL axioms as disjunctive TGDs, we just need unary predicates for concepts, and binary predicates for roles.

A large corpus of works on DLs has focused on the problems of consistency, instance checking and logical entailment. However, the last few years, the attention has shifted on the problem of CQ answering [Ortiz and Simkus 2012], and thus, once again, we are facing the problem of answering CQs under disjunctive TGDs.

1.4. Guardedness and Disjunctive TGDs

It is evident that query answering under disjunctive TGDs is a crucial algorithmic task with several applications. However, as mentioned above, CQ answering under
TGDs is already an undecidable problem [Beeri and Vardi 1984]. A key decidability paradigm, which is decisive for the current work, is guardedness, a well-known restriction that guarantees good model-theoretic and computational properties for first-order sentences. More precisely, the guarded fragment of first-order logic (GFO) was introduced in [Andrěka et al. 1998] with the aim of explaining and generalizing the good properties of modal logic. Guarded formulas are constructed as usual first-order formulas with the exception that all quantification must be bounded, i.e., of the form $\forall X(a \rightarrow \varphi)$ and $\exists X(a \land \varphi)$, where $a$ is an atomic formula that guards $\varphi$ in the sense that it contains all the free variables of $\varphi$. It has been shown that modal logic can be embedded in GFO, and argued in a convincing way that GFO inherits the good properties of modal logic. Furthermore, in [Grädel 1999], it has been established that GFO enjoys the tree-model property, i.e., if a GFO sentence admits a model, then it admits a tree-like model, which in turn implies the decidability of the main reasoning tasks, i.e., satisfiability and query answering. Recently, inspired by GFO, the class of guarded TGDs, that is, TGDs with a guard-atom in the left-hand side that contains (or guards) all the universally quantified variables, has been proposed and investigated [Cali et al. 2013]. Several interesting extensions and restrictions of guarded TGDs have been also considered [Baget et al. 2011a; Cali et al. 2012a]; we refer to all those formalisms by the term guarded-based TGDs.

Guarded-based TGDs can be naturally extended to disjunctive TGDs, without losing the good model-theoretic properties and the decidability of CQ answering. Moreover, the obtained formalisms, apart from the fact that they are theoretically interesting, have several practical applications — observe that all the disjunctive TGDs employed in the Examples 1.1, 1.2 and 1.3 are guarded. In fact, it is not difficult to show that, if a schema mapping specified by source-to-target GAV (global-as-view) TGDs, i.e., TGDs with just one atom in the right-hand side, is quasi-invertible, then it has a quasi-inverse specified by a set of target-to-source guarded disjunctive TGDs; this follows by the definition of the algorithm for producing quasi-inverses given in [Fagin et al. 2008]. Moreover, it is possible to show that the key DL $\text{ALC}$ (end extensions of it) is captured by guarded disjunctive TGDs; more details are given in Section 8. From the above discussion, we understand that guarded-based disjunctive TGDs form a family of first class formalisms with several applications, and thus they deserve our attention.

Although it was known that the problem of CQ answering under guarded-based disjunctive TGDs is decidable (implicit in [Cali et al. 2008]), little was known about its computational complexity before the conference papers [Anonymous I; II; III]. An exception is the work [Alviano et al. 2012], where some special cases of the problem are considered — more details about the complexity results that are inherited from [Alviano et al. 2012] are given in the technical sections. The present paper extends and improves [Anonymous I; II; III] in several ways; more details are given at the end of Section 1.7, where we summarize the contributions of our work.

1.5. Our Main Goal
The main goal of the current work is to better understand the impact of disjunction on query answering under the main guarded-based classes of TGDs. Towards this direction, we perform an in-depth complexity analysis along three different dimensions of the problem under consideration:

(1) **Complexity Type.** Following Vardi’s taxonomy [Vardi 1982], the data complexity of our problem is calculated taking only the database as input, while the query and the set of dependencies are considered fixed. The combined complexity is the complexity calculated considering as input, together with the database, also the query and the set of dependencies. Apart from the combined complexity, we would also
like to understand how the complexity of our problem is affected when some key parameters are fixed. In particular, we consider the following two variants of the combined complexity: (1) the bounded-arity combined complexity, which is calculated by assuming that the arity of the underlying schema is bounded; and (2) the fixed-program combined complexity, which is calculated by considering the set of disjunctive TGDs as fixed (the set of dependencies is usually called program, and hence the term “fixed-program”). Notice that, in practice, the arity of the underlying schema is usually small and can be productively assumed to be fixed. Moreover, the components that change quite often over time are the database and the query, while the program remains the same. Hence, the preceding types of complexity are meaningful metrics that deserve to be investigated.

(2) **Dependency Language.** As already said, we are going to focus on the main guarded-based classes of TGDs extended with disjunction. More precisely, we are going to consider the classes of guarded and frontier-guarded disjunctive TGDs. Recall that guarded disjunctive TGDs have a guard atom in the left-hand side that contains all the universally quantified variables [Cali et al. 2013]. Frontier-guarded disjunctive TGDs extend guarded disjunctive TGDs by requiring in the guard atom only the universally quantified variables that appear also in the right-hand side of the dependency [Baget et al. 2011a]. Each one of the above classes has its weakly counterpart that we would also like to consider in our investigation. Two lightweight formalisms that also deserve our attention are disjunctive inclusion dependencies (IDs), that is, constant-free rules with just one atom in the left- and the right-hand side without repeated variables, and linear disjunctive TGDs. A detailed exposition of all considered formalisms can be found in Section 3.

(3) **Query Language.** The main query language considered in the current study is the language of CQs. A natural extension of CQs, which has been heavily studied in the past, and would be also part of our later complexity analysis, is the class of union of conjunctive queries (UCQs); see, e.g., [Abiteboul et al. 1995]. Several subclasses of (U)CQs have been also considered in the literature with the aim of reducing the complexity of classical database problems such as query evaluation and query containment. The above problems are NP-hard, in general, and become tractable if restricted to one of the following languages: (U)CQs of bounded treewidth [Chekuri and Rajaraman 2000], (U)CQs of bounded hypertree-width [Gottlob et al. 2002], acyclic (U)CQs [Chekuri and Rajaraman 2000], and atomic queries. We would like to understand whether the above subclasses of (U)CQs have an analogous positive impact on query answering under guarded-based disjunctive TGDs.

1.6. **Technical Challenges**

The problem of (U)CQ answering under disjunctive TGDs adheres to the standard logical semantics of entailment (=), which denotes entailment under arbitrary, not necessarily finite, models. This implies that, in general, a database and a set of disjunctive TGDs admit infinitely many models, where each of them can be of infinite size; in fact, this holds already for TGDs due to the existentially quantified variables. Interestingly, in the case of TGDs, it is always possible to construct the so-called universal (a.k.a. canonical) model, which can be seen as a representative of all the other models. Formally, a model of a logical theory is universal if it can be homomorphically embedded into every other model of the theory. Such a universal model of a database and a set of TGDs can be constructed by applying a well-known procedure called the chase (see, e.g., [Johnson and Klug 1984; Fagin et al. 2005; Deutsch et al. 2008]). Roughly, the chase adds new atoms to the given database as dictated by the given TGDs, possibly involving labeled null values as witnesses for the existentially quantified variables,
Guarded-Based Disjunctive Tuple-Generating Dependencies

until the final result satisfies all the TGDs. Therefore, for query answering purposes, instead of considering all the models of a database and a set of TGDs, one can focus on the (possibly infinite) universal model.

The situation changes dramatically if we consider disjunctive TGDs. In fact, there is no single model anymore that acts as a representative of all the other models. However, the notion of the universal model can be naturally extended to the notion of the universal model set. Formally, given a database $D$ and a set $\Sigma$ of disjunctive TGDs, a set $S$ of models of $(D \land \Sigma)$ is called universal if, for each model $M$ of $(D \land \Sigma)$, there exists a model in $S$ that can be homomorphically embedded into $M$. Such a universal model set of a database and a set of disjunctive TGDs can be constructed by applying the so-called disjunctive chase, which is a natural extension of the chase procedure introduced in [Deutsch and Tannen 2003]. However, a universal model set is, in general, infinite, i.e., consists of infinitely many models of infinite size. This is a strong sign that the problem of (U)CQ answering under disjunctive TGDs is more challenging than (U)CQ answering under TGDs, and novel algorithmic techniques must be devised.

1.7. Summary of Contributions

After some technical definitions and preliminaries in Section 2, and the formal definitions of the main guarded-based classes of disjunctive TGDs in Section 3, we proceed with our complexity analysis. In Section 4, we focus on arbitrary (U)CQs, in Section 5 on (U)CQs of bounded treewidth and bounded hypertree-width, in Section 6 on acyclic (U)CQs, and finally, in Section 7, on atomic queries. Each one of the above technical sections starts with an overview subsection, which presents a summary of the obtained results, together with a brief description of the employed techniques, and discusses results inherited from existing works. In what follows, we summarize the general findings of our complexity investigation:

1. **Arbitrary (U)CQs.** For the expressive formalisms, i.e., (weakly-)(frontier-)guarded TGDs, the addition of disjunction does not have a significant effect on the complexity of (U)CQ answering. However, for the less expressive languages, i.e., linear TGDs and inclusion dependencies, the impact can be enormous with, e.g., a jump from NP-completeness to 2\text{ExpTime}-completeness in the case of a fixed set of disjunctive IDs; the latter holds even if we focus on predicates of arity at most three.

2. **(U)CQs of Bounded Treewidth and Hypertree-width.** Queries of bounded treewidth and hypertree-width behave in the same way as arbitrary (U)CQs, and thus they do not have the expected positive impact on the problem under consideration.

3. **Acyclic (U)CQs.** In the case of acyclic (U)CQs, a positive impact can be observed on some relevant settings of the problem. More precisely, for (weakly-)guarded disjunctive TGDs the bounded-arity combined complexity is reduced from 2\text{ExpTime}-complete to \text{ExpTime}-complete, while for (weakly-)frontier-guarded disjunctive TGDs the same can be observed if we focus on fixed programs.

4. **Atomic Queries.** Finally, we observe that atomic queries do not make our problem easier under the classes of disjunctive TGDs that allow more than one atom in the left-hand side of the rules, i.e., (weakly-)(frontier-)guarded disjunctive TGDs. However, the complexity in the case of disjunctive IDs and linear TGDs significantly decreases, and in fact we obtain the first (and, to the best of our knowledge, the only known) tractability results for query answering under disjunctive TGDs. In particular, we obtain a \text{PTime}-membership if we focus on predicates of bounded arity, and \text{AC}^0-membership in data complexity; notice that, for atomic queries, the fixed-program combined complexity and the data complexity coincide.

ACM Transactions on Database Systems, Vol. V, No. N, Article A, Publication date: January YYYY.
Interestingly, our results provide refined lower bounds for the problem of querying the guarded fragment of first-order logic. Moreover, our techniques and results can be used as a generic tool for establishing results on query answering under several central DLs; see Section 8. In particular, we establish two generic complexity results about arbitrary UCQs, UCQs of bounded treewidth, and acyclic UCQs, that can be applied on any DL that is contained in ALCTH, i.e., ALC extended with inverse roles (I) and role hierarchies (H). Those results provide alternative proofs for existing results for ALCTH, as well as novel results for less expressive DLs, such as DL-Lite$^H_{bool}$ [Artale et al. 2009] and ELU [Baader et al. 2005].

As said above, this work is an extended and revised version of the conference papers [Anonymous I; II; III]. A brief comparison between the above papers and the present work follows:

— In [Anonymous I], the complexity of (U)CQ answering focussing on the main guarded-based classes of disjunctive TGDs is investigated. The main result is the 2EXPTIME-hardness of CQ answering under disjunctive IDs, established by a reduction from an appropriate variant of the validity problem of CQs w.r.t. a Büchi automaton [Björklund et al. 2008]. However, the construction is very intricate; in fact, it is a non-trivial adaptation of a 2EXPTIME-hardness proof presented in [Björklund et al. 2008]. In this work, we provide a simpler proof for the 2EXPTIME-hardness of CQ answering under disjunctive IDs by a reduction from the non-acceptance problem of an alternating exponential space Turing machine.

— In [Anonymous II], (U)CQs of bounded treewidth and hypertree-width, as well as acyclic (U)CQs are considered. Although the complete picture for the complexity of the problem under the main guarded-based classes of disjunctive TGDs is presented, no complexity results about DLs are obtained as in [Anonymous I]. This is because the lower bounds established in [Anonymous II] are not strong enough to carry over to DLs. In this work, we present stronger lower bounds for UCQs of bounded treewidth and hypertree-width, and acyclic UCQs, that allow us to establish interesting complexity results about several key DLs.

— In [Anonymous III], the complexity of atomic query answering under guarded and linear disjunctive TGDs is investigated. However, the complexity for (weakly-) frontier-guarded and weakly-guarded disjunctive TGDs, as well as for disjunctive IDs, remained unexplored. In this work, we consider all the above languages, and we present the complete picture for the complexity of atomic query answering.

2. PRELIMINARIES

In this section, we present background material necessary for this paper. We recall some basics on relational databases, (Boolean) conjunctive queries, and disjunctive tuple-generating dependencies. We also introduce the technical tools that we are going to employ in our later technical proofs, namely the disjunctive chase procedure, guarded-based fragments of first-order logic, and some basics on alternating Turing machines. We assume the reader is familiar with fundamental concepts of complexity theory. A detailed exposition of all complexity notions employed in this work can be found in the standard textbooks (see, e.g., [Papadimitriou 1994]).

2.1. General

Let C, N and V be pairwise disjoint infinite countably sets. The elements of C are called constants (constitute the normal domain of a database), the elements of N are called (labeled) nulls (used as placeholders for unknown values, and thus can be also seen as (globally) existentially quantified variables), and the elements of V are called (regular) variables (used in queries and dependencies). A fixed lexicographic order is
assumed on $C \cup N$, such that every value in $N$ follows all those in $C$. We denote by $X$ sequences (or sets) of variables $X_1, \ldots, X_k$. Throughout, let $|n| = \{1, \ldots, n\}$, for every integer $n \geq 1$.

A term $t$ is a constant ($t \in C$), labeled null ($t \in N$), or variable ($t \in V$). An atomic formula (or simply atom) has the form $p(t_1, \ldots, t_n)$, where $p$ is an $n$-ary predicate, and $t_1, \ldots, t_n$ are terms. For an atom $a$, we denote as $\text{dom}(a)$ and $\text{var}(a)$ the set of its terms and the set of its variables, respectively. These notations naturally extend to sets of atoms. For convenience, usually conjunctions and disjunctions of atoms are treated as sets of atoms. An instance $I$ is a (possibly infinite) set of atoms of the form $p(t)$, where $t$ is a tuple of constants and labeled nulls. A database $D$ is a finite instance where only constants occur. Whenever an instance $I$ is treated as a logical formula, in fact it is the formula $\exists X (\bigwedge_{a \in I} a)$, where $X$ contains a variable $X_z$, for each null $z$ in $I$.

A substitution from a set of symbols $S$ to a set of symbols $S'$ is a function $h : S \rightarrow S'$ defined as follows: $\emptyset$ is a substitution (empty substitution), and if $h$ is a substitution, then $h \cup \{ s \rightarrow s' \}$ is a substitution, where $s \in S$ and $s' \in S'$. If $s \rightarrow s' \in h$, then we write $h(s) = s'$. The restriction of $h$ to $T \subseteq S$, denoted $h|_T$, is the substitution $h' = \{ t \rightarrow h(t) \mid t \in T \}$. A homomorphism from a set of atoms $A$ to a set of atoms $A'$ is a substitution $h : C \cup N \cup V \rightarrow C \cup N \cup V$ such that: (i) if $t \in C$, then $h(t) = t$; and (ii) if $p(t_1, \ldots, t_n) \in A$, then $h(p(t_1, \ldots, t_n)) = p(h(t_1), \ldots, h(t_n)) \in A'$.

2.2. Queries

A conjunctive query (CQ) $q$ is a positive existential first-order formula $\exists Y (\varphi(X, Y))$, where $\varphi$ is a conjunction of atoms with variables from $X \cup Y \subset V$, and possibly constants of $C$. The arity of $q$ is defined as the cardinality of $X$, i.e., the number of free variables occurring in $\exists Y (\varphi(X, Y))$. A 0-ary CQ is called Boolean CQ (BCQ). An $n$-ary union of conjunctive queries (UCQ) is a disjunction of a finite number of $n$-ary CQs. By abuse of notation, sometimes we consider a UCQ as set of CQs. The answer to an $n$-ary CQ $q = \exists Y (\varphi(X, Y))$ over an instance $I$, denoted $q(I)$, is the set of all tuples of constants $t \in C^n$ for which there exists a homomorphism $h$ such that $h(\varphi(X, Y)) \subseteq I$ and $h(X) = t$. A BCQ has only the empty tuple as possible answer, in which case it is said to have a positive answer. Formally, a BCQ has a positive answer over $I$, written as $I \models q$, if $q(I) \neq \emptyset$. The answer to an $n$-ary UCQ $Q$ over an instance $I$, denoted $Q(I)$, is the set of $n$-tuples $\bigcup_{q \in Q} q(I)$. The answer to a union of BCQs over $I$ is positive, written as $I \models Q$, if $Q(I) \neq \emptyset$.

2.3. Disjunctive Tuple-generating Dependencies

A disjunctive tuple-generating dependency (DTGD) $\sigma$ is a first-order formula

$$\forall X (\varphi(X) \rightarrow \bigvee_{i=1}^{n} \exists Y_i (\psi_i(X, Y_i))),$$

where $n \geq 1$, $X \cup Y_1 \cup \ldots \cup Y_n \subset V$, and $\varphi, \psi_1, \ldots, \psi_n$ are conjunctions of atoms (possibly with constants of $C$). The formula $\varphi$ is called the body of $\sigma$, denoted body($\sigma$), while $\bigvee_{i=1}^{n} \psi_i$ is the head of $\sigma$, denoted head($\sigma$). The set of variables $\text{var(body(\sigma))} \cap \text{var(head(\sigma))} \subseteq X$, i.e., the variables of $X$ which appear both in the body and in the head of $\sigma$, is known as the frontier of $\sigma$, and is denoted as frontier($\sigma$). If $n = 1$, then $\sigma$ is called tuple-generating dependency (TGD). The schema of a set $\Sigma$ of DTGDs, denoted sch($\Sigma$), is the set of all predicates occurring in $\Sigma$. In the rest of the paper, for brevity, we will omit the universal quantifiers in front of DTGDs, and implicitly assume such a quantification. We will also use the comma (instead of $\wedge$) for conjoining atoms in the body and in the head of a DTGD, and omit the parentheses whenever the scope of an
existential quantifier is clear. An instance $I$ satisfies $\sigma$, written $I \models \sigma$, if the following holds: whenever there exists a homomorphism $h$ such that $h(\varphi(X)) \subseteq I$, then there exists $i \in |n|$ and $h' \supseteq h$ such that $h'(\psi_i(X, Y_i)) \subseteq I$; $I$ satisfies a set $\Sigma$ of DTGDs, denoted $I \models \Sigma$, if $I \models \sigma$, for each $\sigma \in \Sigma$. Whenever a set $\Sigma$ of DTGDs is treated as a logical formula, in fact it is the formula $(\bigwedge_{\sigma \in \Sigma} \sigma)$.

2.4. Query Answering

Given a database $D$ and a set $\Sigma$ of DTGDs, the answers we consider are those that are true in all models of $D$ with respect to $\Sigma$. Formally, the models of $D$ with respect to $\Sigma$, denoted as $\text{mods}(D, \Sigma)$, is the set of all instances $I$ such that $I \supseteq D$ and $I \models \Sigma$. The answer to an $n$-ary CQ $q$ w.r.t. $D$ and $\Sigma$ is the set of $n$-tuples of constants

$$\text{ans}(q, D, \Sigma) = \bigcap_{I \in \text{mods}(D, \Sigma)} q(I).$$

If $q$ is Boolean, then its answer is positive, denoted $D \cup \Sigma \models q$, if $\text{ans}(q, D, \Sigma) \neq \emptyset$. The answer to a UCQ w.r.t. $D$ and $\Sigma$ is defined analogously. The main decision problems tackled in this work are defined as follows:

**Definition 2.1 ((U)CQ Answering).** Given a database $D$, a set $\Sigma$ of DTGDs, an $n$-ary CQ $q$, and a tuple $t \in C^n$, CQ answering is the problem of deciding whether $t \in \text{ans}(q, D, \Sigma)$. The UCQ answering problem is defined analogously. □

It is well-known that CQ answering for arbitrary CQs can be easily reduced to CQ answering for BCQs, just by substituting the given tuple $t$ into the CQ; thus, we can focus on BCQs. Henceforth, by CQ and UCQ we refer to a BCQ and a union of BCQs, respectively. The data complexity of the above problems is calculated taking only the database as input. For the combined complexity, the query and set of DTGDs count as part of the input as well. We assume that the database and the query use only predicates of $\text{sch}(\Sigma)$. Let us explain why this assumption can be made without affecting the generality of our complexity results. Consider a database $D$, a set $\Sigma$ of DTGDs, and a CQ $q$. Let $D = D' \cup D''$, where $\{D', D''\}$ is a partition of $D$ and $D' = \{p(t) \in D \mid p \in \text{sch}(\Sigma)\}$. Moreover, let $q' = \exists X (\varphi(X) \land \psi(X))$, where, for each predicate $p$ occurring in $\varphi(X)$ (resp., $\psi(X)$), $p \in \text{sch}(\Sigma)$ (resp., $p \notin \text{sch}(\Sigma)$). Clearly, if there exists a homomorphism $h$ such that $h(\varphi(X)) \subseteq D''$, then $D \cup \Sigma \models q$ iff $D' \cup \Sigma \models \exists X h(\varphi(X))$; otherwise, $D \cup \Sigma \not\models q$. Notice that CQ answering is at least as hard as checking for the existence of $h$. The above argument can be straightforwardly extended to UCQs.

2.5. Normal Form of DTGDs

For query answering purposes, we can focus on DTGDs that are in normal form. A set $\Sigma$ of DTGDs is in normal form if each $\sigma \in \Sigma$ is of the form

$$\varphi(X) \rightarrow p(X) \quad \text{or} \quad \varphi(X) \rightarrow \exists Y p(X, Y) \quad \text{or} \quad \varphi(X) \rightarrow p_1(X) \lor p_2(X).$$

The first one refers to existential-free single-atom-head TGDs, the second one to single-atom-head TGDs with exactly one occurrence of an existentially quantified variable, while the third one refers to existential-free DTGDs with a disjunction of two atoms in the head. Every set $\Sigma$ of DTGDs can be transformed in logarithmic space into a set $N(\Sigma)$ in normal form such that, for every database $D$ and UCQ $Q$, $D \cup \Sigma \models Q$ iff $D \cup N(\Sigma) \models Q$. The normalization procedure can be found in Appendix A.

In the sequel, in order to simplify our technical proofs, sometimes we will focus on normalized sets of DTGDs. It will be explicitly stated when the DTGDs under consideration are assumed to be in normal form.
2.6. Disjunctive Chase

In our investigation of (U)CQ answering, we employ the disjunctive chase introduced in [Deutsch and Tannen 2003], that is, an extension of the well-known chase procedure. Each disjunctive chase step "branches" out several instances, each satisfying one of the disjuncts of the DTGD that is applied, and thus the result of the disjunctive chase is, in general, a set of instances (not a single instance as that of the classical chase). The disjunctive chase works on an instance through the so-called DTGD chase rule:

Definition 2.2. (Chase Rule). Consider an instance \( I \), and a DTGD \( \sigma \) of the form \( \varphi(X) \rightarrow \bigvee_{i=1}^{n} \exists Y \psi_i(X, Y) \). \( \sigma \) is applicable to \( I \) if there exists a homomorphism \( h \) such that \( h(\varphi(X)) \subseteq I \), and the result of applying \( \sigma \) to \( I \) with \( h \) is the set \{\( I_1, \ldots, I_n \)\}, where \( I_i = I \cup h'(\psi_i(X, Y)) \), for each \( i \in [n] \), and \( h' \supseteq h \) is such that \( h'(Y) \) is a "fresh" null not occurring in \( I \), and following lexicographically all those in \( I \), for each \( Y \in Y \). For such an application, which defines a single chase step, we write \( I(\sigma, h)\{I_1, \ldots, I_n\} \).

A disjunctive chase tree of a database \( D \) and a set \( \Sigma \) of DTGDs is a (possibly infinite) tree such that the root is \( D \), and for every node \( I \), assuming that \{\( I_1, \ldots, I_n \)\} are the children of \( I \), there exists \( \sigma \in \Sigma \) and a homomorphism \( h \) such that \( I(\sigma, h)\{I_1, \ldots, I_n\} \). The disjunctive chase algorithm for \( D \) and \( \Sigma \) consists of an exhaustive application of DTGD chase steps in a fair fashion, which leads to a disjunctive chase tree \( T \) of \( D \) and \( \Sigma \); let \( \text{chase}(D, \Sigma) \) be the set \{\( I \mid I \) is a leaf of \( T \)\}. Notice that each leaf of \( T \) is well-defined as the least fixpoint of a monotonic operator. By construction, each instance of \( \text{chase}(D, \Sigma) \) is a model of \( D \) and \( \Sigma \). Interestingly, \( \text{chase}(D, \Sigma) \) is a universal model set of \( D \) and \( \Sigma \), i.e., for each \( I \in \text{mods}(D, \Sigma) \), there exists \( J \in \text{chase}(D, \Sigma) \) and a homomorphism \( h_J \) such that \( h_J(J) \subseteq I \) [Deutsch et al. 2008]. This universality property implies the following useful result, which actually shows that the disjunctive chase is a formal algorithmic tool for query answering purposes.

Theorem 2.3. Consider a database \( D \), a set \( \Sigma \) of DTGDs, and a UCQ \( Q \). It holds that \( D \cup \Sigma \models Q \) iff \( I \models Q \), for each \( I \in \text{chase}(D, \Sigma) \).

2.7. Guarded-based Fragments of First-order Logic

In our later complexity analysis, we will exploit complexity results established for expressive guarded-based fragments of first-order logic.

Guarded Fragment. The guarded fragment of first-order logic (GFO), introduced in [Andrěka et al. 1998], is the collection of first-order formulas with some syntactic restrictions in the quantification pattern, which is analogous to the relativised nature of modal logic. The set of GFO formulas over a schema \( \mathcal{R} \) is the smallest set

\( (1) \) containing all atomic \( \mathcal{R} \)-formulas and equalities;
\( (2) \) closed under the logical connectives \(-, \land, \lor, \rightarrow\); and
\( (3) \) if \( a \) is an \( \mathcal{R} \)-atom or an equality atom containing all the variables of \( X \cup Y \), and \( \varphi \) is a GFO formula with free variables contained in \( (X \cup Y) \), then
\[ \forall X(a \rightarrow \varphi) \quad \text{and} \quad \exists X(a \land \varphi) \]
are GFO formulas as well.

The loosely guarded fragment of first-order logic (LGFO) is a generalization of GFO where the quantifiers are guarded by conjunctions of atomic formulas such that the quantifier-rule (3) is relaxed as follows:

\( (3) \) if \( a_1 \land \ldots \land a_n \) is a conjunction of \( \mathcal{R} \)-atoms containing all the variables of \( X \cup Y \), for every pair of variables \( (V, W) \in X \times (X \cup Y) \) there exists at least one atom

of \{a_1, \ldots, a_n\} that contains both V and W, and \(\varphi\) is an LGFO formula with free variables contained in \((X \cup Y)\), then
\[
\forall X((a_1 \land \ldots \land a_n) \rightarrow \varphi) \quad \text{and} \quad \exists X((a_1 \land \ldots \land a_n) \land \varphi)
\]
are LGFO formulas as well.

**Guarded Negation.** Guarded negation first-order logic (GNFO) restricts first-order logic by requiring that all occurrences of negation are of the form \(a \land \neg \varphi\), where \(a\) is an atom containing all the free variables of \(\varphi\) [Bárány et al. 2011]. Formally, the formulas of GNFO are generated by the recursive definition
\[
\varphi ::= p(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2 \mid \exists X \varphi \mid a \land \neg \varphi,
\]
where each \(t_i \in C \cup V\), and in the last clause, \(a\) is an atomic formula containing all free variables of \(\varphi\). Notice that GNFO is strictly more expressive than GFO and incomparable to LGFO.

### 2.8. Alternating Turing Machines

An alternating Turing machine is a tuple \(M = (S, \Lambda, \delta, s_0)\), where \(S = S_v \cup S_e \cup \{s_a\} \cup \{s_r\}\) is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, \(\Lambda\) is the tape alphabet, \(\delta \subseteq (S \times \Lambda) \times (S \times \Lambda \times \{-1, 0, +1\})\) is the transition relation, and \(s_0 \in S\) is the initial state. We assume that \(\Lambda\) contains a special blank symbol \(\sqcup\). The symbols \(-1, 0\) and \(+1\) denote the cursor directions left, stay and right.

A computation tree for \(M\) is a tree labelled by configurations (tape content, cursor position, and internal state) of \(M\) such that (1) if node \(v\) is labelled by an existential configuration, then \(v\) has one child, labelled by one of the possible successor configurations; (2) if \(v\) is labelled by a universal configuration, then \(v\) has one child for each possible successor configuration; (3) the root is labelled by the initial configuration; and (4) all leaves are labelled by accepting or rejecting configurations. A computation tree is accepting if it is finite and all leaves are labelled by accepting configurations.

### 3. GUARDED-BASED CLASSES OF DTGDs

(U)CQ answering under (non-disjunctive) TGDs is undecidable [Beeri and Vardi 1981], even when the set of TGDs is fixed [Cali et al. 2013], or even when the set of TGDs is singleton [Baget et al. 2011a]. Several syntactic restrictions can be found in the literature that guarantee the decidability of (U)CQ answering under TGDs. The property which is of special interest for the current work is guardedness. In particular, guardedness ensures that the instance constructed by the chase procedure is tree-like, which in turn implies the decidability of (U)CQ answering. In this section, we introduce the disjunctive version of the main guarded-based classes of TGDs that can be found in the literature. Then, we show that the tree-likeness of the chase is preserved, and thus, as we shall see, also the decidability of the problems under consideration is preserved.

Before we proceed further, let us recall the notion of affected positions, introduced in [Cali et al. 2013] in the context of TGDs, and can be naturally defined also for DTGDs. Given an \(n\)-ary predicate \(p\), the position \(p[i]\) is identified by \(p\) and its \(i\)-th attribute. We refer to the arity of \(p\) by \(arity(p)\). Given a set \(\Sigma\) of DTGDs, the set of positions of \(sch(\Sigma)\), denoted \(pos(\Sigma)\), is the set \(\{p[i] \mid p \in sch(\Sigma)\ \text{and} \ i \in \{arity(p)\}\}\). Intuitively, a position \(\pi \in pos(\Sigma)\) is affected if, during the construction of the disjunctive chase w.r.t. \(\Sigma\), \(\pi\) may host a null value of N. Formally, the set of affected positions of \(\Sigma\), denoted \(affected(\Sigma)\), is inductively defined as follows: (1) if there exists \(\sigma \in \Sigma\) such that at position \(\pi\) an existentially quantified variable occurs, then \(\pi \in affected(\Sigma)\); and (2) if there
exists $\sigma \in \Sigma$ and a variable $X$ that occurs in $\text{body}(\sigma)$ only at positions of $\text{affected}(\Sigma)$, and $X$ appears in $\text{head}(\sigma)$ at position $\pi$, then $\pi \in \text{affected}(\Sigma)$.

**Example 3.1 (Affected Positions).** Consider the set $\Sigma$ consisting of the DTGDs:

\[
\begin{align*}
\sigma_1 &: t(X, Y), s(Y) \rightarrow \exists Z t(Y, Z) \\
\sigma_2 &: t(X, Y) \rightarrow p(Y) \\
\sigma_3 &: p(X) \rightarrow p_1(X) \lor p_2(X).
\end{align*}
\]

Clearly, $t[2] \in \text{affected}(\Sigma)$ due to the existentially quantified variable occurring at $t[2]$ in $\text{head}(\sigma_1)$. Since the variable $Y$ occurs at position $t[2]$ in $\text{body}(\sigma_2)$ and at position $p[1]$ in $\text{head}(\sigma_2)$, we conclude that $p[1] \in \text{affected}(\Sigma)$. Finally, due to $\sigma_3$, both $p[1]$ and $p[2]$ belong to $\text{affected}(\Sigma)$. The position $t[1]$ is not affected w.r.t. $\Sigma$ since, although the same variable $Y$ occurs at $t[2] \in \text{affected}(\Sigma)$ in $\text{body}(\sigma_1)$ and at $t[1]$ in $\text{head}(\sigma_1)$, $Y$ occurs also at the non-affected position $s[1]$.

Having the above auxiliary notion in place, we are now ready to define the main guarded-based classes of DTGDs.

**Definition 3.2 (Classes of DTGDs).** Fix a DTGD $\sigma$.

- $\sigma$ is weakly-frontier-guarded (resp., weakly-guarded) w.r.t. a set $\Sigma$ of DTGDs if there exists an atom $a \in \text{body}(\sigma)$ which contains (or guards) all the variables of $\text{frontier}(\sigma)$ (resp., $\text{var}(\text{body}(\sigma))$) that appear only at positions of $\text{affected}(\Sigma)$ — the set $\Sigma$ is called weakly-(frontier-)guarded if, for every $\sigma' \in \Sigma$, $\sigma'$ is weakly-(frontier-)guarded w.r.t. $\Sigma$;
- $\sigma$ is frontier-guarded (resp., guarded) if there exists an atom $a \in \text{body}(\sigma)$ which contains all the variables of $\text{frontier}(\sigma)$ (resp., $\text{var}(\text{body}(\sigma))$);
- $\sigma$ is multi-linear if every atom $a \in \text{body}(\sigma)$ contains all the variables of $\text{var}(\text{body}(\sigma))$;
- $\sigma$ is linear if $|\text{body}(\sigma)| = 1$, i.e., $\sigma$ has only one body-atom; and
- $\sigma$ is a disjunctive inclusion dependency (DID) if it is constant-free, $|\text{body}(\sigma)| = 1$, $\text{head}(\sigma)$ is a disjunction of atoms, each $X \in \text{var}(\text{body}(\sigma))$ occurs at most once in $\text{body}(\sigma)$, and for each atom $a \in \text{head}(\sigma)$, each $X \in \text{var}(a)$ occurs at most once in $a$. $\square$

Notice that the above classes of DTGDs are closed under the normalization procedure introduced in Section 2, i.e., for a set $\Sigma$ of DTGDs which falls in a class $C$, where $C$ is one of the classes introduced in Definition 3.2, $\Sigma(\Sigma)$ also falls in $C$.

**Decidability of UCQ Answering.** We conclude this section by showing that indeed UCQ answering under weakly-frontier-guarded sets of DTGDs (and thus, under all the other classes of DTGDs introduced above) is decidable. We show this by exploiting the classical result that fragments of first-order logic which enjoy the finite treewidth model property are decidable [Courcelle 1989]². The treewidth of a graph is a well-known notion in graph theory, which measures how similar a graph is to a tree.

**Definition 3.3 (Treewidth).** Given a graph $G = (V, E)$, a tree decomposition of $G$ is a pair $(T, \lambda)$, where $T = (N, A)$ is a tree, and $\lambda$ is a labeling function $N \rightarrow 2^V$ such that:

1. for all $v \in V$, there exists $u \in N$ such that $v \in \lambda(u)$;
2. for all edges $(v_1, v_2) \in E$, there exists $u \in N$ such that $\{v_1, v_2\} \subseteq \lambda(u)$; and
3. for every $v \in V$, the set $\{u \mid v \in \lambda(u)\} \subseteq N$ induces a connected subtree of $T$.

The width of $(T, \lambda)$ is $\max_{u \in N}(|\lambda(u)|) - 1$. The treewidth of a graph $G$, denoted as $tw(G)$, is the minimum width among all possible tree decompositions of $G$. $\square$

²In fact, this result was shown for fragments of monadic second-order logic.

ACM Transactions on Database Systems, Vol. V, No. N, Article A, Publication date: January YYYY.
Let us now recall the treewidth of an instance. The hypergraph of an instance \( I \), denote \( H(I) \), is a hypergraph \((V, H)\), where \( V = \text{dom}(I) \), and, for each \( g \in I \), there exists a hyperedge \( h \in H \) such that \( h = \text{dom}(g) \). The treewidth of an instance \( I \), denoted \( \text{tw}(I) \), is defined as the treewidth of its hypergraph \( H(I) \), that is, the treewidth of the Gaifman graph (a.k.a. primal graph) \( G_{H(I)} \) of \( H(I) \). Recall that the Gaifman graph of \( H(I) \) is the graph \((V, E)\), where \( V \) is the node set of \( H(I) \), and \((v, u) \in E \) iff \( H(I) \) has a hyperedge \( h \) such that \( \{v, u\} \subseteq h \).

Example 3.4 (Treewidth of an Instance). Consider the instance

\[
I = \{p(a, b, c), p(c, f, d), p(e, f, g), t(b, c), t(b, e), t(e, f), t(f, c)\},
\]

where \( \{a, b, c, d, e, f, g\} \subseteq C \). The hypergraph \( H(I) \) is depicted in Figure 1(a), its Gaifman graph \( G_{H(I)} \) in Figure 1(b), and a tree decomposition of \( G_{H(I)} \) (c).

It is easy to verify that the given tree decomposition is optimal, and thus \( \text{tw}(I) = 2 \).

A fragment \( \mathcal{L} \) of first-order logic enjoys the finite treewidth model property if the following holds: for each formula \( \varphi \in \mathcal{L} \), whenever \( \varphi \) is satisfiable, then there exists a model of \( \varphi \) (which can be seen as a relational instance) of finite treewidth. Let us now give our decidability result:

**Theorem 3.5.** UCQ answering under weakly-frontier-guarded DTGDs is decidable.

**Proof.** Let \( \mathcal{L}_{WFG} \) be the fragment of first-order logic which can express only formulas of the form \( \Phi_{D, \Sigma, Q} = (D \land \Sigma \land \neg Q) \), where \( D \) is a database, \( \Sigma \) is a weakly-frontier-guarded set of DTGDs, and \( Q \) is a UCQ. Since \( D \cup \Sigma \models Q \) iff \( \Phi_{D, \Sigma, Q} \) is unsatisfiable, it suffices to show that \( \mathcal{L}_{WFG} \) enjoys the finite treewidth model property. Consider an arbitrary formula \( \Phi_{D, \Sigma, Q} \in \mathcal{L}_{WFG} \). By the universality property of the disjunctive chase, \( D \cup \Sigma \models Q \) iff \( D \models Q \), for each \( I \in \text{chase}(D, \Sigma) \). Hence, \( \Phi_{D, \Sigma, Q} \) is satisfiable iff there exists \( I \in \text{chase}(D, \Sigma) \) such that \( (I \land \neg Q) \) is satisfiable. Clearly, if \( \Phi_{D, \Sigma, Q} \) is satisfiable, then there exists \( I \in \text{chase}(D, \Sigma) \) which is a model of \( \Phi_{D, \Sigma, Q} \). By definition of the disjunctive chase, the instance \( I \) can be seen as the result of the (non-disjunctive) chase under \( D \) and a set of weakly-frontier-guarded (non-disjunctive) TGDs. It is implicit in [Baget et al. 2011a], where weakly-frontier-guarded (non-disjunctive) TGDs are investigated, that \( I \) has finite treewidth, and the claim follows.
Table I: Complexity of answering (U)CQs under guarded-based DTGDs.

<table>
<thead>
<tr>
<th>Combined Complexity</th>
<th>Bounded Arity</th>
<th>Fixed Theory</th>
<th>Data Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DID</td>
<td>2ExpTime</td>
<td>2ExpTime</td>
<td>coNP</td>
</tr>
<tr>
<td></td>
<td>LB: Thm. 4.8</td>
<td>LB: [Calvanese et al. 2013, Thm. 4.5]</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>2ExpTime</td>
<td>2ExpTime</td>
<td>coNP</td>
</tr>
<tr>
<td>ML</td>
<td>2ExpTime</td>
<td>2ExpTime</td>
<td>coNP</td>
</tr>
<tr>
<td>G</td>
<td>2ExpTime</td>
<td>2ExpTime</td>
<td>coNP</td>
</tr>
<tr>
<td>FG</td>
<td>2ExpTime</td>
<td>2ExpTime</td>
<td>coNP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LB: [Cali et al. 2013, Thm. 4.1]</td>
<td></td>
</tr>
<tr>
<td>WG</td>
<td>2ExpTime</td>
<td>2ExpTime</td>
<td>ExpTime</td>
</tr>
<tr>
<td>WFG</td>
<td>2ExpTime</td>
<td>2ExpTime</td>
<td>ExpTime</td>
</tr>
<tr>
<td></td>
<td>UB: Thm. 4.1</td>
<td>UB: Thm. 4.4</td>
<td></td>
</tr>
</tbody>
</table>

The above theorem establishes decidability of (U)CQ answering under the guarded-based classes of DTGDs that we consider in this work. However, it tells nothing about the computational complexity of our problem. Understanding the complexity of the problem under consideration will be the subject of the next sections.

4. ARBITRARY QUERIES

In this section, we focus on answering (unions of) conjunctive queries under our respective classes of DTGDs. The overview section below contains a summary of our results, and discusses results inherited from existing works. Full proofs for our novel upper and lower complexity bounds are given in Section 4.2 and 4.3, respectively.

4.1. Overview

Table I summarizes the complexity results for answering (U)CQs under the various DTGD formalisms considered in this paper. Each row corresponds to a class of DTGDs (which is decoded by substituting L for linear, G for guarded, F for frontier, and W for weakly), while each column corresponds to a different setting of the problem. In each cell of the table, we have indicated where to find the corresponding results; UB and LB stands for upper and lower bound, respectively. Note that missing references for the upper (resp., lower) bounds are immediately inherited from the first lower-left (resp., upper-right) cell in which a reference is given. The results of this section, together with a brief description of the employed techniques, follow:

Upper Bounds:

1. UCQ answering under weakly-frontier-guarded DTGDs is in 2ExpTime in combined complexity (Theorem 4.1) — this is established by a reduction to the unsatisfiability problem of GNFO sentences, which is 2ExpTime-complete [Bárány et al. 2011, Theorem 2];
(2) UCQ answering under frontier-guarded DTGDs is in \text{coNP} in data complexity (Theorem 4.3) — this is proved by a reduction to UCQ answering under GFO, which is \text{coNP}-complete in data complexity [Barany et al. 2010, Theorem 19]; and
(3) UCQ answering under weakly-frontier-guarded DTGDs is in \text{EXPTIME} in data complexity (Theorem 4.4) — this is shown by a reduction to UCQ answering under GFO sentences, and then exploit the fact the latter is feasible in exponential time in the size of the sentence [Barany et al. 2010, Theorem 19].

Lower Bounds:

(1) UCQ answering under DIDs is \text{2ExpTime}-hard in combined complexity, even if we focus on predicates of arity at most two (Theorem 4.5) — this is established by a reduction from the non-acceptance problem of an alternating exponential space Turing machine;
(2) UCQ answering under a fixed set of DIDs is \text{2ExpTime}-hard, even if we consider predicates of arity at most two (Theorem 4.6) — this is shown by adapting the reduction in the proof of Theorem 4.5 in such a way that the constructed set of DTGDs does not depend on the Turing machine; and
(3) CQ answering under a fixed set of DIDs is \text{2ExpTime}-hard, even if we consider predicates of arity at most three (Theorem 4.8) — we first show that UCQ answering under (arbitrary) DTGDs is reducible in polynomial time to CQ answering at the price of increasing the arity of the underlying schema by one (Lemma 4.7), and then we exploit Theorem 4.6.

Clearly, the \text{2ExpTime} upper bound for weakly-frontier-guarded DTGDs (the most expressive formalism considered here), and the \text{2ExpTime} lower bound in the case of a fixed set of DIDs (the weakest formalism studied in this paper), close the picture of the computational complexity of our problem w.r.t. the combined complexity, the case of bounded arity, and the case of a fixed set of DTGDs. Existing results provide us with two optimal lower bounds for the data complexity of our problem, which complete the entire picture of the complexity of (U)CQ answering.

Inherited Results:

(1) CQ answering under DIDs is \text{coNP}-hard in data complexity [Calvanese et al. 2013, Theorem 4.5] — in fact, Theorem 4.5 in [Calvanese et al. 2013] shows that CQ answering under a TBox with a single description logic axiom of the form $A_1 \sqsubseteq A_2 \sqcup A_3$, where each $A_i$ is an atomic concept, which in turn is logically equivalent to the DID $A_1(X) \rightarrow A_2(X) \lor A_3(X)$, is \text{coNP}-hard in data complexity; and
(2) CQ answering under weakly-guarded DTGDs is \text{ExpTime}-hard in data complexity [Cali et al. 2013, Theorem 4.1] — actually, Theorem 4.1 in [Cali et al. 2013] considers weakly-guarded TGDs (without disjunctive heads).

From the results of this section, we observe that for the expressive formalisms under consideration, namely weakly-(frontier-)guarded DTGDs, the addition of disjunction does not have a significant effect on the complexity of (U)CQ answering. However, for the less expressive formalisms, the impact can be enormous with, e.g., a jump from \text{NP}-completeness to \text{2ExpTime}-completeness in the case of a fixed set of DIDs. Let us now proceed with the formal proofs of our results.

4.2. Upper Bounds

We start this section by showing the following result:

\begin{theorem} \textit{UCQ answering under weakly-frontier-guarded DTGDs is in \text{2ExpTime} in combined complexity.} \end{theorem}
GUARDED-BASED DISJUNCTIVE TUPLE-GENERATING DEPENDENCIES
A:17

PROOF (SKETCH). We provide a polynomial-time reduction to the problem of deciding whether a GFO sentence is unsatisfiable, which in turn is in 2EXPSPACE [Barany et al. 2011]. Consider a database \( D \), a set \( \Sigma \) of weakly-frontier-guarded DTGDs, and a UCQ \( Q \). We construct a database \( D' \), a set \( \Sigma' \) of frontier-guarded DTGDs, and a UCQ \( Q' \), such that \( D \cup \Sigma \models Q \) iff \( D' \cup \Sigma' \models Q' \); for the construction see Appendix B. Now, observe that a frontier-guarded DTGD \( \sigma \) of the form \( \forall X (\varphi(X) \rightarrow \exists Y \psi(X,Y)) \) can be equivalently rewritten as the sentence \( \Phi_{\sigma} = \neg(\exists Y (\varphi(X) \land \neg\exists Y \psi(X,Y))) \), which falls in GFO since all the free variables of \( \exists Y \psi(X,Y) \) appear in the frontier-guard of \( \varphi(X) \). Moreover, given a (Boolean) CQ \( q \), \( \neg q \) trivially falls in GFO since in \( q \) there are no free variables. From the above discussion, we conclude that the sentence \( \Psi_{D',\Sigma',Q'} = (D' \land \bigwedge_{\sigma \in \Sigma'} \Phi_{\sigma} \land \bigwedge_{q \in Q} \neg q) \) falls in GNFO. The claim follows since \( D' \cup \Sigma' \models Q' \) iff \( \Psi_{D',\Sigma',Q'} \) is unsatisfiable.

Notice that an alternative way to obtain the above result is to reduce our problem to UCQ answering under GFO sentences, which is also in 2EXPSPACE [Barany et al. 2010]. However, the translation from frontier-guarded DTGDs to GFO is straightforward, whereas the translation to GFO is more involved, as GFO imposes tighter syntactic restrictions on formulas.

We now focus on the data complexity of UCQ answering under (weakly-)frontier-guarded DTGDs. It is known that CQ answering under frontier-guarded TGDs can be reduced in linear time to UCQ answering under GFO sentences [Baget et al. 2011]. Interestingly, the same reduction (with minor adaptations), that can be found in Appendix B, can be employed even if we consider frontier-guarded DTGDs:

**Lemma 4.2.** UCQ answering under frontier-guarded DTGDs can be reduced in linear time to UCQ answering under GFO sentences.

By exploiting the above auxiliary result, we are now ready to establish the desired upper bounds for the data complexity of our problem.

**Theorem 4.3.** UCQ answering under frontier-guarded DTGDs is in coNP in data complexity.

**Proof.** The result follows from Lemma 4.2, and the fact that UCQ answering under GFO sentences is in coNP in data complexity [Barany et al. 2010].

We now focus on weakly-frontier-guarded DTGDs:

**Theorem 4.4.** UCQ answering under weakly-frontier-guarded DTGDs is in \( \text{XP} \) in data complexity.

**Proof.** Consider a database \( D \), a set \( \Sigma \) of weakly-frontier-guarded DTGDs, and a UCQ \( Q \). We first reduce our problem to UCQ answering under frontier-guarded DTGDs by replacing the non-affected variables in the DTGDs of \( \Sigma \) with all possible constants occurring in \( D \). In other words, we partially ground the set \( \Sigma \), and we obtain a set \( \Sigma' \) of frontier-guarded DTGDs. Clearly, \( \Sigma' \) is of exponential size in the number of non-affected variables, but of polynomial size in \(|\text{dom}(D)|\). By Lemma 4.2, there exists a linear translation \( \tau \) such that \( D \cup \Sigma' \models Q \) iff \( D \cup \tau(\Sigma') \models \tau(Q) \), where \( \tau(\Sigma') \) is a GFO sentence and \( \tau(Q) \) a UCQ. It is important to say that, although \(|\tau(Q)| \) depends on \( D \), the size of each CQ in \( \tau(Q) \) does not depend on \( D \). As shown in [Barany et al. 2010], UCQ answering under GFO sentences depends doubly-exponentially on the size of each CQ of the given UCQ and the maximum arity of the schema, and exponentially on the size of the given GFO sentence. Since the size of each query of \( \tau(Q) \) and the maximum arity of the schema are constant in \( D \), while the size of \( \tau(\Sigma') \) is polynomial in \( D \), we get an \( \text{XP} \) upper bound w.r.t. \( D \), and the claim follows. 

ACM Transactions on Database Systems, Vol. V, No. N, Article A, Publication date: January YYYY.
Recall that UCQ answering under DIDs is coNP-hard in data complexity [Calvanese et al. 2013], while for weakly-guarded DTGDs is EXPTIME-hard [Cali et al. 2013]. Thus, the picture of the data complexity of CQ answering under our guarded-based classes of DTGDs is now complete.

4.3. Lower Bounds

We now present a series of strong 2EXPTIME lower bounds for answering arbitrary (U)CQs under DIDs. The general idea is to force a (binary) tree structure to appear in every model of the disjunctive chase, representing a sequence of configurations. Each such configuration is in turn represented by a full binary tree of depth $n$, such that the $2^n$ leafs represent the tape contents, as well as the cursor position and state, of the simulated Turing machine. We will then construct a UCQ, where each disjunct checks if there is an error in the tree structure (i.e., if it does not represent a valid computation of the Turing machine).

**Theorem 4.5.** UCQ answering under DIDs is 2EXPTIME-hard in combined complexity, even for predicates of arity at most two.

**Proof.** The proof is via a reduction from the non-acceptance problem of an alternating exponential space Turing machine $M$ on the empty input. Let $M = (S, \Lambda, \delta, s_0)$, where $S = S_\forall \cup S_\exists \cup \{s_a\} \cup \{s_r\}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, $\Lambda = \{0, 1, \sqcup\}$ is the tape alphabet with $\sqcup$ be the blank symbol, $\delta : S \times \Lambda \rightarrow (S \times \Lambda \times \{-1, +1\})^2$ is the transition function, and $s_0 \in S$ is the initial state. We assume that $M$ is well-behaved and never tries to read beyond its tape boundaries, always halts, and uses exactly $2^n$ tape cells, where $n > 1$. Furthermore, we assume that a rejecting configuration does not have a subsequent configuration, while an accepting configuration has only itself as a subsequent configuration. We also assume that $s_0 \in S_\exists$, and also that every universal configuration is followed by two existential configurations and vice versa (note that this allows us to represent the transition relation as a function, as above). Finally, for technical reasons, which will be clarified later, we assume that $M$ never touches the first and the last cells of the tape, and also starts with its head at the second cell. The above assumptions can be made, without sacrificing the generality of our proof, since the non-acceptance problem of $M$ remains 2EXPTIME-hard. Our goal is to construct a database $D$, a set $\Sigma$ of DTGDs, and a UCQ $Q$ such that $D \cup \Sigma \models Q$ iff $M$ rejects, $\text{N}(\Sigma)$ is a set of DIDs, and $\text{sch}(\text{N}(\Sigma))$ consists of unary and binary predicates.

The general idea underlying our proof is to construct trees, which encode possible computation trees of $M$, by chasing $D$ and $\Sigma$, and then check their consistency via the
Each configuration has two successor configurations, such that a universal configuration is followed by existential configurations, and vice-versa. Note that for existential configurations, both are forced to appear in the same model:

\[ conf_{\exists}(X) \rightarrow \exists Y \exists Y child(X, Y), conf_{\forall}(Y), \]
\[ conf_{\forall}(X) \rightarrow \exists Y \exists Z child(X, Y), conf_{\exists}(Y), \]
\[ child_{i}(X, Y) \rightarrow child(X, Y), \text{ for each } i \in \{1, 2\}. \]

Roughly speaking, the atom \( child(z_1, z_2) \) expresses the fact that \( z_1 \) is a configuration with \( z_2 \) be one of its subsequent configurations.

Each configuration has a configuration tree, i.e., a full binary tree of depth \( n \) with \( 2^n \) leaf nodes, which simulates its tape and encodes its state:

\[ conf_x(X) \rightarrow level_0(X), \text{ for each } x \in \{\exists, \forall\}, \]
\[ level_i(X) \rightarrow \exists Y \exists Y child_{L}(X, Y_L), level_{i+1}(Y_L), \]
\[ child_{R}(X, Y_R), level_{i+1}(Y_R), \text{ for each } i \in [n - 1], \]
\[ child_{2}(X, Y) \rightarrow child(X, Y), \text{ for each } x \in \{L, R\}, \]
\[ level_n(X) \rightarrow \bigvee_{\lambda \in \Lambda} \lambda(X), \]
\[ level_n(X) \rightarrow cursor(X) \lor notcursor(X), \]
\[ cursor(X) \rightarrow \bigvee_{s \in S} s(X). \]

Notice that the last three DTGDs guess the content of the tape cells (recall that each leaf node of the configuration tree represents a cell), as well as the state of the machine and the position of the cursor.

In order for our UCQ to be able to navigate through the constructed tree, we attach a navigation gadget to each node of a configuration tree, using the following TGDs:

\[ child_{L}(X, Y) \rightarrow \exists Y_1 \exists Y_2 \exists Y_3 child(Y_1, Y_2), child(Y_1, Y_3), 0(Y_1), p(Y_2), 1(Y_3), \]
\[ child_{R}(X, Y) \rightarrow \exists Y_1 \exists Y_2 \exists Y_3 child(Y_1, Y_2), child(Y_1, Y_3), 1(Y_1), p(Y_2), 0(Y_3), \]
\[ child(X, Y) \rightarrow parentorchild(X, Y), \]
\[ child(X, Y) \rightarrow parentorchild(Y, X). \]
The navigation gadget encodes whether a particular node is the left or right child of its parent. Left children have a chain of length three attached to them, where the first node is labeled by 0, the second by p, and the third by 1. For right children, the labels 0 and 1 are reversed. An example of a navigation gadget, which encodes the left (resp., right) child of a node, is depicted in Figure 3(a) (resp., 3(b)).

The construction of Σ is now complete. It is easy to verify that N(Σ) is a set of DIDs, and also that sch(N(Σ)) consists of unary and binary predicates.

The UCQ Q. We now proceed with the construction of Q, which ensures that each model of D ∪ Σ encodes an invalid computation tree (and thus D ∪ Σ |= Q iff M rejects). Q consists of the following disjuncts:

1. Q_{initial} for checking that in the initial configuration is invalid, i.e., that the cursor is not at the second tape cell; the second tape cell is labelled with anything but the starting state s₀; the third checks that some tape cell contains a non-blank symbol; and the fourth checks that the cursor is at some tape cell except the second one (i.e., that there are multiple cursor positions).

The sequel, for a binary predicate r, we write r^i(X, Y), where i > 0, as a shorthand for ∃Z_1...∃Z_{i-1} (r(X, Z_1) ∧ ... ∧ r(Z_{i-1}, Y)); for i ≤ 0, r^i(X, Y) is defined as X = Y. Note that our definition of CQs does not include equalities, however, we can simply remove the equality by replacing all occurrences of X with Y. Henceforth, we will interpret an atom X = Y in this way. Let us now formalize Q.

1. Q_{initial} contains four disjuncts. The first checks whether the cursor is not at the second tape cell; the second checks whether the second tape cell is labelled with anything but the starting state s₀; the third checks that some tape cell contains a non-blank symbol; and the fourth checks that the cursor is at some tape cell except the second one (i.e., that there are multiple cursor positions). Q_{initial} is defined as follows; recall that c ∈ C represents the initial configuration of M:

\[ \exists X \exists Y \left( \text{child}_{n-1}^L(c, X) \land \text{child}_R(X, Y) \land \text{nocursor}(Y) \right) \lor \ \exists X \exists Y \left( \text{child}_{n-1}^R(c, X) \land \text{child}_R(X, Y) \land s(Y) \right) \lor \ \exists X \exists Y \left( \text{child}^n(c, X) \land \lambda(X) \right) \lor \ \exists X \exists Y \exists Z \left( \text{child}_1^L(c, X) \land \text{child}_R^{min(1,n-i)}(X, Y) \land \text{child}^{n-i-1}(Y, Z) \land \text{cursor}(Z) \right) \]
Before giving the definition of \( Q_{\text{trans}} \) and \( Q_{\text{inertia}} \), we first need to define two auxiliary subqueries:

— \( \text{siblings}(X, Y, Z) \), which is true for three leaf nodes \( X, Y \) and \( Z \) iff they represent three subsequent tape cells in the same configuration; and

— \( \text{sameCell}(X, Y) \), which is true for two leaf nodes \( X \) and \( Y \) iff they represent the same tape cell and they belong to successive configurations, i.e., if they belong to configurations \( C_X \) and \( C_Y \), respectively, then \( C_Y \) is a successor configuration of \( C_X \).

In the definition of \( \text{siblings} \), we exploit the following key property, which can be easily verified, that two neighboring leaves \( v_1 \) and \( v_2 \) in a full binary tree, where \( v_1 \) appears to the left of \( v_2 \), fulfill: there exists a common ancestor node \( v \) such that \( v_1 \) (resp., \( v_2 \)) can be reached from \( v \) by a path that leads to the left (resp., right) once, and only to the right (resp., left) afterwards. The subquery \( \text{siblings} \) is defined as follows:

\[
\begin{align*}
\text{siblings}(X, Y, Z) & \equiv \exists P_1 \exists P_2 \exists P_3 \exists P_4 \left( \text{child}_L(P_1, X) \land \text{child}_R(P_1, Y) \land \text{child}_L(P_2, P_3) \land \text{child}_R(P_2, P_4) \land \\
& \quad \bigvee_{1 < i < n} \left( \text{child}_R(P_3, Y) \land \text{child}_L(P_4, Z) \right) \right) \lor \\
& \exists P_1 \exists P_2 \exists P_3 \exists P_4 \left( \text{child}_L(P_1, P_2) \land \text{child}_R(P_1, Z) \land \text{child}_L(P_2, P_3) \land \text{child}_R(P_2, P_4) \land \\
& \quad \bigvee_{1 < i < n} \left( \text{child}_R(P_3, X) \land \text{child}_L(P_4, Y) \right) \right).
\end{align*}
\]

Note that the above query contains a mixture of disjunctions and conjunctions, and will be used as a conjunct in subsequent queries, yielding a query that is not strictly a UCQ. However, by unfolding via distribution, a UCQ of at most quadratic size can be obtained. Any use as the above should be seen as such an unfolded UCQ.

We proceed now with the definition of \( \text{sameCell} \). In fact, \( \text{sameCell} \) comes in two varieties: \( \text{sameCell}_1(X, Y) \) and \( \text{sameCell}_2(X, Y) \) depending on whether \( Y \) is in the first or second successor configuration from \( X \). Let us for the moment assume that we can call a subquery \( \Psi_i(X, Y) \), which is expresses that if \( X \) and \( Y \) are nodes at level \( i > 0 \) in successive configuration trees, then \( \Psi_i(X, Y) \) is true iff \( X \) and \( Y \) are both either left or right children of their parent nodes. Then, for each \( j \in [2] \),

\[
\begin{align*}
\text{sameCell}_j(X, Y) & \equiv \exists X_0 \ldots \exists X_{n-1} \exists Y_0 \ldots \exists Y_{n-1} \\
& \quad \left( \text{level}_0(X_0) \land \text{level}_0(Y_0) \land \text{child}_j(X_0, Y_0) \land \\
& \quad \quad \bigwedge_{1 \leq i \leq n-1} \left( \text{child}(X_{i-1}, X_i) \land \text{child}(Y_{i-1}, Y_i) \land \Psi_i(X_i, Y_i) \right) \land \\
& \quad \quad \text{child}(X_{n-1}, X) \land \text{child}(Y_{n-1}, Y) \land \Psi_i(X, Y) \right).
\end{align*}
\]

We now define the subquery \( \Psi_i(X, Y) \). It verifies, for two nodes \( X \) and \( Y \) at level \( i \) in subsequent configuration trees, whether \( X \) and \( Y \) are both left children or both right children of their parents. Notice that left and right children can be distinguished by the position of their \( l \)-labeled navigation gadget node relative to their \( p \)-labeled navigation gadget node. In particular, a node \( v \) at level \( i \) of a configuration tree and a node \( u \) at level \( i \) of a successor configuration tree are both left or both right children, if the nodes \( v_1 \) and \( u_1 \) with label 1 in their respective gadgets have a common ancestor that has distance \( i+3 \) from \( v_1 \) and \( i+4 \) from \( u_1 \). Note that for the first and second configuration in our tree, this common ancestor actually lies before the starting configuration marked
by the constant $c$; hence, the two ancestor nodes $a$ and $b$ introduced in the database $D$. The definition of $\Psi_i(X, Y)$ follows, where variable $U$ denotes the common ancestor:

$$
\Psi_i(X, Y) = \exists P_X \exists P_Y \exists T_X \exists T_Y \exists U \ (p(P_X) \land p(P_Y) \land 1(T_X) \land 1(T_Y) \land \text{child}^2(X, P_X) \land \text{child}^2(Y, P_Y)
\text{parentorchild}(P_X, T_X) \land \text{parentorchild}(P_Y, T_Y) \land \text{child}^{i+3}(U, T_X) \land \text{child}^{i+4}(U, T_Y)).
$$

We are now ready to define $Q_{\text{trans}}$ and $Q_{\text{inertia}}$.

(2) For the definition of $Q_{\text{trans}}$, it is more convenient to consider the transitions of $\delta$ as rewriting assertions of the form

$$(x, y, z) \rightarrow ((x_1, y_1, z_1), (x_2, y_2, z_2)),$$

where $x, z \in \Lambda, y \in (\Lambda \times \Lambda), x_i, z_i \in (\Lambda \cup (\Lambda \times \Lambda))$ and $y_i \in \Lambda$, for each $i \in [2]$. Moreover, for each $i \in [2], x_i \in \Lambda$ implies $z_i \in (\Lambda \times \Lambda)$ and $x_i \in (\Lambda \times \Lambda)$ implies $y_i \in \Lambda$. Intuitively, $y$ represents the state of the machine and the symbol read by the cursor, while $x$ and $z$ are the symbols to the left and right of the cursor position. In a subsequent configuration such a partial configuration $(x,y,z)$ then transforms into $(x_1,y_1,z_1)$ and $(x_2,y_2,z_2)$, according to $\delta$. Before defining $Q_{\text{trans}}$, we need to introduce some auxiliary notions. First, we define the complement of $\delta$ relative to universal and existential states, denoted $\delta_{\forall}$ and $\delta_{\exists}$, respectively. Formally, $\delta_{\forall}: S \times \Lambda \rightarrow (S \times \Lambda \times \{-1, +1\})^2$ consists of all the transition rules of the form $(s, a) \rightarrow ((s_1, a_1, d_1), (s_2, a_2, d_2))$, where $s \in S_{\forall}$, not occurring in $\delta$. Analogously, $\delta_{\exists}: S \times \Lambda \rightarrow (S \times \Lambda \times \{-1, +1\})^2$ consists of all the transition rules of the form $(s, a) \rightarrow ((s_1, a_1, d_1), (s_2, a_2, d_2))$, where $s \in S_{\exists}$, for which there is no transition rule of the form $(s, a) \rightarrow ((s_1, a_1, d_1), (-))$ or $(s, a) \rightarrow (\cdot, (s_2, a_2, d_2))$ occurring in $\delta$. Let $T_{\forall} = \bigcup_{\tau \in \delta_{\forall}} f(\tau)$ and $T_{\exists} = \bigcup_{\tau \in \delta_{\exists}} f(\tau)$, where, assuming that $\tau = (s, a) \rightarrow ((s_1, a_1, d_1), (s_2, a_2, d_2))$,

$$
f(\tau) = \begin{cases} 
\bigcup_{x,y \in \Lambda} \{(x, (s, a), y) \rightarrow ((x, a_1, (s_1, y)), (x, a_2, (s_2, y)))\}, & \text{if } d_1 = +1, d_2 = +1 \\
\bigcup_{x,y \in \Lambda} \{(x, (s, a), y) \rightarrow ((x, a_1, (s_1, y)), ((s_2, a_2, y)), a_2)), & \text{if } d_1 = +1, d_2 = -1, \\
\bigcup_{x,y \in \Lambda} \{(x, (s, a), y) \rightarrow (((s_1, x, a_1, y), (x, a_2, (s_2, y))\}, & \text{if } d_1 = -1, d_2 = +1, \\
\bigcup_{x,y \in \Lambda} \{(x, (s, a), y) \rightarrow (((s_1, x, a_1, y), ((s_2, a_2, y)), a_2)), & \text{if } d_1 = -1, d_2 = -1. 
\end{cases}
$$

In the definition of $Q_{\text{trans}}$, we exploit the binary operator $\odot^s$, where $s \in S$, which is defined as $\lor$, if $s \in S_2$, and as $\land$, if $s \in S_{\forall}$. We also make use of the shorthand $\text{is}_x(\cdot) \equiv a(\cdot)$, if $x = a \in \Lambda$, and $\text{is}_x(\cdot) \equiv a(\cdot) \land s(\cdot)$, if $x = (s, a) \in (\Lambda \times \Lambda)$. $Q_{\text{trans}}$ is defined as follows:

$$
\bigvee_{(x,y)=(s,a) \rightarrow ((x_1,y_1,z_1),(x_2,y_2,z_2)) \in (T_{\forall} \cup T_{\exists}) \in \{1,2\}} \exists X \exists Y \exists X_L \exists X_R \exists Y_L \exists Y_R
\text{sameCell}_i(X, Y) \land \text{siblings}(X_L, X, X_R) \land \text{siblings}(Y_L, Y, Y_R) \land
\text{is}_x(X_L) \land \text{is}_y(X) \land \text{is}_z(X_R) \land \text{is}_x(Y_L) \land \text{is}_y(Y) \land \text{is}_z(Y_R).
$$

(3) Finally, $Q_{\text{inertia}}$ is defined as

$$
\bigvee_{(a,a') \in \Lambda \times \Lambda, a \neq a'} \exists X \exists Y \exists X_L \exists X_R \exists Y_L \exists Y_R
\text{sameCell}_i(X, Y) \land \text{siblings}(X_L, X, X_R) \land \text{siblings}(Y_L, Y, Y_R) \land
\text{notcursor}(X) \land a(X) \land a'(Y).
$$
Notice that, by employing the above query, we do not guarantee the inertia of the first and the last cells of the tape. This is not a problem since, by assumption, $M$ never touches those cells, and also starts with its cursor at the second cell.

The definition of $Q$ is now complete. It is easy to verify that $Q$ is a UCQ.

By construction, and also the assumption that a rejecting configuration does not have a subsequent configuration, but an accepting one has only itself as a subsequent configuration, we get that $D \cup \Sigma \models Q$ iff $M$ rejects. More precisely, if $M$ accepts, then there exists $I \in \text{chase}(D, \Sigma)$ that encodes an accepting computation tree of $M$. Thus, $I \not\models Q$, which implies that $D \cup \Sigma \not\models Q$. Conversely, if $M$ rejects, then, for each $I \in \text{chase}(D, \Sigma)$, $I \models Q_{\text{trans}}$. Towards a contradiction, assume that $M$ rejects and there exists $I \in \text{chase}(D, \Sigma)$ such that $I \not\models Q_{\text{trans}}$. By construction, $I$ encodes an infinite tree of configurations. But since a rejecting configuration does not have a subsequent configuration (by assumption), necessarily at least one disjunct of $Q_{\text{trans}}$ will be entailed by $I$ which is a contradiction. Therefore, $D \cup \Sigma \models Q$, and the claim follows.

Notice that the constructed set $\Sigma$ of DIDs in the above proof depends on the Turing machine $M$ that we encode. In fact, the part of $\Sigma$ that depends on $M$ are the DIDs that construct the configuration trees — observe that we use $n$ level-predicates, one for each level of the tree, and $|S|$ state-predicates, one for each state of $M$. Interestingly, a new construction can be devised in such a way that $\Sigma$ is independent from $M$, and thus it is fixed. In order to avoid to use $n$ level-predicates, we can employ a single unary predicate, called $\text{ctnode}$, which stores all the nodes of a configuration tree. In order to avoid to use $|S|$ state-predicates, we can replace each state by a chain of length at most $|S|$; notice that such chains can easily be constructed via two simple DIDs. Then, we can access a state by exploiting those chains, without explicitly encode them inside a predicate (as we did in the proof of Theorem 4.5). After implementing this idea, the set of DIDs no longer depends on the Turing machine, and thus it is fixed as desired. However, we need to guarantee that each chain encodes a valid state, which can be checked by adding another disjunct in the UCQ $Q$; for the formal proof see Appendix B.

**Theorem 4.6.** UCQ answering under fixed sets of DIDs is $2\text{EXPTIME}$-hard, even for predicates of arity at most two.

The question that comes up is whether the above hardness result can be established for CQs. Interestingly, this question is answered positively, at the expense of increasing the arity of the underlying schema by one. This is done by first showing that there exists a polynomial time reduction of UCQ answering under DTGDs (not only DIDs) to CQ answering under DTGDs, which increases the arity of the underlying schema by one. Before giving the formal result, let us briefly explain how such a reduction works; the formal construction can be found in Appendix B. Each predicate $p$ of the underlying schema is replaced by a new predicate $p'$ with arity $\text{arity}(p) + 1$. This extra position holds a marker, either $t$ (for $\text{true}$) or $f$ (for $\text{false}$). Every database atom gets the value $t$ at this new position, which implies that is a valid atom. Moreover, each DTGD is extended so that it simply propagates this position unaltered to the head. A copy of each disjunct (i.e., CQ) of the given UCQ is added to the database (variables are replaced by new distinct constants) with $f$ at the new position. Clearly, every CQ in the given UCQ trivially maps to the database, with $f$ at the new position. However, all the valid atoms, i.e., the atoms that can be derived by the chase of the original database w.r.t. the original set of DTGDs, have $t$ at the new position. We can now replace disjunctions in the UCQ by conjunctions (and thus convert the UCQ into a CQ), and just check that, for at least one query, its new position maps to $t$. This can
simply be done by adding to the database atoms of the form \( \text{or}(\cdot,\cdot,\cdot) \) encoding the logical \( \text{or} \), connecting all the subqueries of the original UCQ via such \( \text{or} \) predicates, and stating that the end result must be \( t \). The formal result follows:

**Lemma 4.7.** Consider a database \( D \), a set \( \Sigma \) of DTGDs, and a UCQ \( Q \). A database \( D' \), a set \( \Sigma' \) of DTGDs, where \( \text{arity}(\text{sch}(\Sigma')) = \text{arity}(\text{sch}(\Sigma)) + 1 \), and a CQ \( q \) can be constructed in polynomial time such that \( D \cup \Sigma \models Q \) iff \( D' \cup \Sigma' \models q \).

It is important to say that the construction employed in the proof of the above lemma preserves the syntactic properties of all classes of DTGDs considered in this paper. Moreover, \( \Sigma' \) does not depend on the database or the query, but only on the set \( \Sigma \) of DTGDs. Therefore, Theorem 4.6 and Lemma 4.7 imply the desired result:

**Theorem 4.8.** CQ answering under fixed sets of DIDs is \( 2\text{ExpTime}\)-hard, even for predicates of arity at most three.

Recall that UCQ answering under weakly-frontier-guarded DTGDs is in \( 2\text{ExpTime} \)-TIMe in combined complexity (Theorem 4.1). Hence, the picture of the computational complexity of CQ answering under our guarded-based classes of DTGDs is now complete. Interestingly, Theorem 4.8 closes an open question stated in [Barany et al. 2010], regarding the complexity of query answering under fixed GFO sentences. It was shown that the problem in question is already \( \text{PSPACE}\)-hard for CQs, and in \( \text{ExpTime} \) in case of a restricted class of CQs. However, the exact complexity was left as an open problem. Clearly, the above result gives a \( 2\text{ExpTime}\)-completeness result since query answering under GFO is in \( 2\text{ExpTime} \) in general.

**Corollary 4.9.** (U)CQ answering under fixed GFO sentences is \( 2\text{ExpTime}\)-complete.

5. BOUNDED TREEWIDTH QUERIES

Several subclasses of CQs have been considered in the literature, with the aim of reducing the complexity of classical database problems such query evaluation and query containment. Such a well-known fragment is the class of CQs of bounded treewidth (BTWCQs), i.e., the treewidth of their hypergraph is bounded by an integer constant; see, e.g., [Chekuri and Rajaraman 2000]. Let us recall that the hypergraph \( \mathcal{H}(q) \) of a query \( q \) is defined in the same way as the hypergraph \( \mathcal{H}(I) \) of an instance \( I \) (see Section 3). Our goal in this section is to understand whether the complexity of (U)CQ answering under our guarded-based classes of DTGDs is affected if we focus on queries of bounded treewidth.

Unfortunately, queries of bounded treewidth do not have the expected positive impact on the complexity of our problem. Table I, which summarizes the complexity for answering (U)CQs, holds even if we consider BTW(U)CQs. As you can observe, the data complexity for all the classes of DTGDs under consideration is obtained from existing results. More precisely, the coNP-hardness for DID is obtained from [Calvanese et al. 2013, Theorem 4.5], where it is shown that CQ answering under a single DID of the form \( p_1(X) \rightarrow p_2(X) \lor p_3(X) \), is already coNP-hard, even if the input query is fixed (and thus of bounded treewidth). The coNP upper bound for frontier-guarded DTGDs follows from Theorem 4.3. The \( \text{ExpTime}\)-hardness for weakly-guarded DTGDs is inherited from [Cali et al. 2013, Theorem 4.1], where it is shown that CQ answering under a fixed set of weakly-guarded TGDs is \( \text{ExpTime}\)-hard, even if the input query is a single atom. The \( \text{ExpTime}\) upper bound for weakly-frontier-guarded DTGDs follows from Theorem 4.4.

Although the data complexity of our problem can be settled by exploiting known results, this is not the case for the other settings of the problem. The best known upper
Fig. 4: The hypergraph \( H(q) \).

bound is the 2\text{EXPTIME} upper bound for answering arbitrary UCQs under weakly-frontier-guarded DTGDs (Theorem 4.1). This result, combined with the fact that CQ answering under guarded TGDs is 2\text{EXPTIME}-hard in the combined complexity [Cali et al. 2013, Theorem 6.1], even for atomic queries of the form \( \exists X p(X) \) (and thus of bounded treewidth), closes the combined complexity picture of our problem. Interestingly, the queries employed in the proofs of Theorems 4.5, 4.6 and 4.8, are already of bounded treewidth. This immediately implies that the above theorems hold also for (U)CQs of bounded treewidth, and the next result follows:

\textbf{Theorem 5.1.} BTW(U)CQ answering under fixed sets of DIDs is 2\text{EXPTIME}-hard, even for predicates of arity at most three (two).

Another key class of queries is the class of CQs of bounded hypertree-width [Gottlob et al. 2002]. The hypertree-width is a measure of how close to acyclic a hypergraph is, analogous to treewidth for graphs. The hypertree-width of a CQ is less than or equal to its treewidth. Since all the upper bounds in Table I hold for arbitrary (U)CQs, we get that arbitrary (U)CQs, (U)CQs of bounded treewidth and (U)CQs of bounded hypertree-width are indistinguishable w.r.t. the complexity of query answering under our guarded-based classes of DTGDs.

6. ACYCLIC QUERIES

In this section, we focus on another important subclass of conjunctive queries, which has been proposed with the aim of reducing the complexity of query evaluation and query containment, namely the class of acyclic CQs; see, e.g., [Chekuri and Rajaraman 2000]. Our goal is to understand how the complexity of (U)CQ answering under our guarded-based classes of DTGDs is affected if we focus on acyclic queries. The acyclicity of a CQ \( q \) is defined via the acyclicity of its hypergraph \( H(q) \). Thus, a crucial notion in the definition of acyclic CQs is the acyclicity of a hypergraph. Informally, a hypergraph is acyclic if it can be reduced to the empty hypergraph by iteratively eliminating some non-maximal hyperedge, or some vertex contained in at most one hyperedge; this procedure is known as the GYO algorithm. The formal definition is as follows.

Consider a hypergraph \( H = (V, H) \). A hyperedge \( h \in H \) is called an ear if (1) \( h \cap h' = \emptyset \), for each \( h' \in H \setminus \{h\} \), or (2) there exists \( h' \in H \setminus \{h\} \) such that \( h \cap (\bigcup_{h'' \in H \setminus \{h \}} h'') \subseteq h' \). The \textit{GYO-reduct} of \( H \), denoted \( \text{GYO}(H) \), is obtained by applying exhaustively the following two steps until \( H \) has no ears: (1) choose an arbitrary ear \( h \) of \( H \); and (2) let \( H' = (V', H \setminus \{h\}) \), where \( V' = \bigcup_{h' \in H \setminus \{h\}} h' \). We say that \( H \) is acyclic if \( \text{GYO}(H) = (\emptyset, \emptyset) \). A CQ \( q \) is acyclic (ACQ) if \( H(q) \) is acyclic.

\textbf{Example 6.1 (Acyclic Query).} Consider the CQ

\[ q : \exists A_1 \ldots \exists A_6 p(A_1, A_2) \land p(A_3, A_4) \land s(A_1, A_4, A_5) \land s(A_4, A_5, A_6). \]
Table II: Complexity of answering acyclic (U)CQs under guarded-based DTGDs.

<table>
<thead>
<tr>
<th></th>
<th>Combined Complexity</th>
<th>Bounded Arity</th>
<th>Fixed Theory</th>
<th>Data Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DID</strong></td>
<td>2EXPTime</td>
<td>EXPTime</td>
<td>EXPTime</td>
<td>coNP</td>
</tr>
<tr>
<td></td>
<td>LB: Thm. 6.4</td>
<td></td>
<td>LB: Thm. 6.6</td>
<td>[Calvanese et al. 2013, Thm. 4.5]</td>
</tr>
<tr>
<td><strong>L</strong></td>
<td>2EXPTime</td>
<td>EXPTime</td>
<td>EXPTime</td>
<td>coNP</td>
</tr>
<tr>
<td><strong>ML</strong></td>
<td>2EXPTime</td>
<td>EXPTime</td>
<td>EXPTime</td>
<td>coNP</td>
</tr>
<tr>
<td><strong>G</strong></td>
<td>2EXPTime</td>
<td>EXPTime</td>
<td>EXPTime</td>
<td>coNP</td>
</tr>
<tr>
<td><strong>FG</strong></td>
<td>2EXPTime</td>
<td>2EXPTime</td>
<td>EXPTime</td>
<td>coNP</td>
</tr>
<tr>
<td></td>
<td>LB: Thm. 6.5</td>
<td></td>
<td></td>
<td>UB: Thm. 4.3</td>
</tr>
<tr>
<td><strong>WG</strong></td>
<td>2EXPTime</td>
<td>EXPTime</td>
<td>EXPTime</td>
<td>EXPTime</td>
</tr>
<tr>
<td></td>
<td>UB: Thm. 6.2</td>
<td></td>
<td></td>
<td>LB: [Cal et al. 2013, Thm. 4.1]</td>
</tr>
<tr>
<td><strong>WFG</strong></td>
<td>2EXPTime</td>
<td>2EXPTime</td>
<td>EXPTime</td>
<td>EXPTime</td>
</tr>
<tr>
<td></td>
<td>UB: Thm. 4.1</td>
<td></td>
<td></td>
<td>UB: Thm. 4.4</td>
</tr>
</tbody>
</table>

The hypergraph $\mathcal{H}(q)$ is shown in Figure 4. Clearly, the hyperedge $\{A_4, A_5, A_6\}$ is an ear, and thus can be eliminated. Now, both $\{A_1, A_2\}$ and $\{A_3, A_4\}$ are ears and can be eliminated. Finally, the remaining hyperedge $\{A_1, A_4, A_5\}$ is an ear, since it does not intersect with any other hyperedge, and can be eliminated. Thus, $GYO(\mathcal{H}(q)) = (\emptyset, \emptyset)$, which means that $q$ is acyclic.

We start by summarizing our results, and then proceed with our new upper and lower complexity bounds in Section 6.2 and 6.3, respectively.

6.1. Overview

Table II summarizes the complexity results for answering (U)CQs under the various DTGD formalisms considered in this paper. Compared with the results in Table I, it is immediately apparent that the combined and the data complexity do not change if we restrict ourselves to acyclic queries. The complexity decreases from 2EXPTime to EXPTime for the less expressive classes of DTGDs, namely DIDs, (multi-)linear and guarded, in the case of predicates of bounded arity, and also for all the classes if we consider a fixed set of DTGDs. The novel results of this section follow:

Upper Bounds:

(1) AUCQ answering under weakly-guarded sets of DTGDs is in EXPTime if we focus on predicates of bounded arity (Theorem 6.2) — this is shown by a reduction to satisfiability of the loosely-guarded fragment of first-order logic, and then exploit the fact that the latter is in EXPTime in case of predicates of bounded arity [Grädel 1999]; and
(2) AUCQ answering under fixed weakly-frontier-guarded sets of DTGDs is in EXPTime (Theorem 6.3) — by a reduction to UCQ answering under (non-fixed) GFO
sentences with predicates of bounded arity, where in the constructed (partially acyclic) query the cyclic part is fixed.

Lower Bounds:

(1) ACQ answering under DIDs is $2\text{ExpTime}$-hard in combined complexity (Theorem 6.4) — this is established by simulating the behavior of an alternating exponential space Turing machine by means of a set of DIDs and an ACQ;

(2) ACQ answering under frontier-guarded DTGDs is $2\text{ExpTime}$-hard, even if we consider predicates of bounded arity (Theorem 6.5) — this is shown by a reduction from CQ answering under frontier-guarded DTGDs; and

(3) ACQ answering under fixed sets of DIDs is $\text{ExpTime}$-hard (Theorem 6.6) — again by simulating the behavior of an alternating linear space Turing machine.

The $\text{ExpTime}$ upper bound for guarded DTGDs in the case of predicates of bounded arity, and for fixed sets of weakly-frontier-guarded DTGDs, and the $\text{ExpTime}$ lower bound for fixed sets of DIDs, close the picture of the computational complexity of our problem, for DIDs, (multi-)linear and guarded in the case of bounded arity, and for all classes in the case of a fixed set of DTGDs. The missing upper bounds are inherited from results of Section 4; for details see Table II. Finally, the following results from the literature provide us with optimal lower bounds for the data complexity of our problem, and the entire picture of the complexity of A(U)CQ answering is completed.

Inherited Results:

(1) ACQ answering under DID is coNP-hard in data complexity [Calvanese et al. 2013, Theorem 4.5] — the CQ employed in the proof of this result is of the form $\exists X \exists Y_1 \exists Z_1 (\bigwedge_{i \in \{1,2\}} (p_i(X, Y_i) \land s(Y_i) \land r_i(X, Z_i) \land t(Z_i)))$, which is clearly acyclic; and

(2) ACQ answering under weakly-guarded DTGDs is $\text{ExpTime}$-hard in data complexity [Cali et al. 2013, Theorem 4.1] — the CQ used in the proof of this result consists of a single atom, and thus is trivially acyclic.

From the results of this section, we observe that the acyclicity of the query does not make the query answering problem easier w.r.t. the combined and data complexity. However, in the cases of bounded arity (for the less expressive classes) and fixed sets of DTGDs, the complexity decreases from $2\text{ExpTime}$ to $\text{ExpTime}$. Let us now proceed with the formal proofs of our results.

6.2. Upper Bounds

We start this section by showing the following:

**Theorem 6.2.** AUCQ answering under weakly-guarded sets of DTGDs is in $\text{ExpTime}$ in case of predicates of bounded arity.

**Proof (Sketch).** Consider a database $D$, a weakly-guarded set $\Sigma$ of DTGDs with predicates of bounded arity, and an AUCQ $Q$. The proof is by reduction to satisfiability of LGFO sentences. More precisely, we construct, in polynomial time, a set $\Sigma'$ of DTGDs such that $\Psi_{D,\Sigma',Q} = (D \land \Sigma' \land \neg Q)$ falls in LGFO, the arity of the underlying schema is increased by two (and therefore is bounded), and $D \cup \Sigma \models Q$ iff $\Psi_{D,\Sigma',Q}$ is unsatisfiable. Since the problem of deciding whether an LGFO sentence is in $\text{ExpTime}$ in case of predicates of bounded arity [Grädel 1999], the claim follows. The formal proof can be found in Appendix C.

ACM Transactions on Database Systems, Vol. V, No. N, Article A, Publication date: January YYYY.
Although the above result does not hold for weakly-frontier-guarded DTGDs, the next theorem shows that we get an \textsc{ExpTime} upper bound if we focus on fixed weakly-frontier-guarded sets of DTGDs.

**Theorem 6.3.** \textsc{Aucq} answering under fixed weakly-frontier-guarded sets of DTGDs is in \textsc{ExpTime}.

**Proof.** Consider a database $D$, a fixed weakly-frontier-guarded set of DTGDs, and an \textsc{Aucq} $Q$. First, we reduce our problem to UCQ answering under GFO sentences in the usual way: (1) partially ground the DTGDs of $\Sigma$ in such a way that the non-affected variables are replaced by all possible combinations of constants of $\text{dom}(D)$, yielding an equivalent set of frontier-guarded DTGDs $\Sigma'$; and (2) in linear time reduce the problem of deciding whether $D \cup \Sigma' \models Q$ to the problem of deciding whether the GFO formula $(D \land \Psi_{\Sigma'})$ entails the UCQ $(Q \lor \Psi_{\Sigma'})$ by employing Lemma 4.2. Since $\Sigma$ is fixed, we immediately get that $\Sigma'$ is of polynomial size w.r.t. $|\text{dom}(D)|$. Thus, by construction, $(D \land \Psi_{\Sigma'})$ and $(Q \lor \Psi_{\Sigma'})$ are of polynomial size, and they use only predicates of bounded arity. Before we proceed further, we need to recall some details regarding the problem of UCQ answering under GFO as presented in [Barany et al. 2010]. Given a GFO formula $\varphi$ and a UCQ $Q'$, the way that the problem of deciding whether $\varphi \models Q'$ is tackled is as follows: (1) $Q'$ is rewritten as a GFO formula $\chi(Q')$ (this procedure is called treeification) such that $\varphi \models Q'$ iff $\varphi \models \chi(Q')$; and (2) it is checked whether the GFO formula $(\varphi \land \neg \chi(Q'))$ is unsatisfiable. The latter check is feasible in time $2^{O(n + |\varphi| + |\chi(Q')| \cdot m^m)}$, where $|\cdot|$ denotes the size of a formula, $n$ is the number of predicates occurring in $(\varphi \land \neg \chi(Q'))$, and $m$ is the maximum arity over all these predicates. Let us now explain how the desired upper bound is obtained. Clearly, it suffices to check whether the formula $(D \land \Psi_{\Sigma'} \land \neg \chi(Q) \land \neg \chi(\Psi_{\Sigma'})$ is of polynomial size, and also the maximum arity of the underlying schema is bounded. It remains to show that $|\chi(Q)|$ and $|\chi(\Psi_{\Sigma'})|$ are of polynomial size. Since $Q$ is acyclic, we know that the treeification of $Q$ is feasible in polynomial time [Barany et al. 2010], and thus $|\chi(Q)|$ is polynomial. By construction, since the set $\Sigma$ of DTGDs is fixed, we get that $|\Psi_{\Sigma'}|$ is constant. Hence, by definition of the treeification procedure, $|\chi(\Psi_{\Sigma'})|$ is constant, and the claim follows.

This concludes the upper bound section for acyclic queries.

**6.3. Lower Bounds**

We start this section by showing that ACQ answering under DID$s$ is 2\textsc{ExpTime}-hard. The construction is rather tedious, and hinges on the fact that, using a polynomial number of DIDs, it is possible to generate and store an exponential number of integers in binary form that can be used to index the exponentially many tape cells of an alternating \textsc{ExpSpace} Turing machine. Then, the computation of such a machine can be simulated by means of a set of DIDs and a CQ, and the 2\textsc{ExpTime}-hardness result follows from the fact that alternating \textsc{ExpSpace} coincides with 2\textsc{ExpTime}.

**Theorem 6.4.** \textsc{Acq} answering under DIDs is 2\textsc{ExpTime}-hard in combined complexity.

**Proof.** The proof is by a reduction from the non-acceptance problem of an alternating exponential space Turing machine $M$ on the empty input. Let $M = (S, \Lambda, \delta, s_0)$, where $S = S_U \uplus S_\Sigma \uplus \{s_0\} \uplus \{s_r\}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, $\Lambda = \{0, 1, \sqcup\}$ is the tape alphabet with $\sqcup$ be the blank symbol, $\delta : S \times \Lambda \rightarrow (S \times \Lambda \times \{-1, +1\})^2$ is the transition function, and $s_0 \in S$ is the initial state. We assume that $M$ is well-behaved and
never tries to read beyond its tape boundaries, always halts, and uses exactly $2^n$ tape cells, where $n > 1$. Furthermore, we assume that a rejecting configuration does not have a subsequent configuration, while an accepting configuration has only itself as a subsequent configuration. Finally, we assume that $s_0 \in S_3$, and also that every universal configuration is followed by two existential configurations and vice versa. The above assumptions can be made, without sacrificing the generality of our proof, since the non-acceptance problem of $M$ remains 2EXPTIME-hard. Our goal is to construct a database $D$, a set $\Sigma$ of DTGDs, and an AUCQ $Q$ such that $D \cup \Sigma \models Q$ iff $M$ rejects, and $N(\Sigma)$ is a set of DIDs. By exploiting the construction in the proof of Lemma 4.7, a database $D'$, a set $\Sigma'$, and an ACQ $q$ can be constructed in polynomial time such that $D \cup \Sigma \models Q$ iff $D' \cup \Sigma' \models q$, and $N(\Sigma')$ is a set of DIDs, which proves the claim. We proceed with the construction of $D$, $\Sigma$ and $Q$.

The general idea of the proof is to construct trees, which encode possible computation trees of $M$, by chasing $D$ with $\Sigma$, and then use the query $Q$ to check the consistency of those trees. In order to represent configurations of $M$, we will use atoms of the form $conf[s](b_1, \ldots, b_n, a, h, t, p, n_1, n_2)$, where $s \in S$ is the state of the encoded configuration and is part of the predicate, $(b_1, \ldots, b_n) \in \{0, 1\}^n$ is an integer of $\{0, \ldots, 2^n - 1\}$ in binary encoding that represents the index of the encoded cell with $a$ be its content, $h \in \{0, 1\}$ and $h = 1$ iff the cursor of $M$ is at the encoded cell, and $t, p, n_1$ and $n_2$ represent the current ($t$ for this), the previous and the next two configurations, respectively. For example, assuming that $n = 3$, $conf[s](1, 0, 1, \sqcup, 1, z_2, z_1, z_3, z_4)$, where $\{z_1, \ldots, z_4\} \subset \mathbb{N}$, says that the state of the configuration $z_2$ is $s$, the fifth cell contains the blank symbol, the cursor is at the fifth cell, the previous configuration of $z_2$ is $z_1$, and the next two configurations of $z_2$ are $z_3$ and $z_4$.

The Database $D$. Let the database $D$ contain a single atom $begin(0, 1, \sqcup, 0^{2^n}, 1^n, 1, c)$; for a constant $x$, $x^k$ denotes the sequence $x, \ldots, x$ with $k$ occurrences of $x$. The first three constants $0, 1$ and $\sqcup$ will allow us to have access to the symbols of $\Lambda$ without explicitly mention them in the DTGDs; recall that DIDs are constant-free. The $2n$ zeros and $n$ ones are auxiliary constants that will allow us to generate, for each configuration of $M$, the $2^n$ integers in binary that will be used to index the tape cells that $M$ can touch. The constant 1 will be used to ensure that exactly one tape cell is pointed by the cursor. Finally, $c$ is a constant that represents the initial configuration.

The Set $\Sigma$. We proceed now with the construction of $\Sigma$. In the sequel, for brevity, we write $X^k$ for the sequence of variables $X_1, \ldots, X_k$.

— We first construct the initial configuration with the TGD

$$begin(Z, O, B, Z^{2^n}, O^n, H, T) \rightarrow \exists P \exists N_1 \exists N_2 \text{conf}_0[s_0](Z, O, B, Z^{2^n}, O^n, H, T, P, N_1, N_2).$$

Notice that in the head of the above TGD we use the auxiliary predicate $\text{conf}_0[s]$, where $s \in S$, and not the predicate $\text{conf}[s]$. This is because we first want to generate all the possible trees that may encode a computation tree of $M$, where the nodes are labeled with auxiliary atoms of the form $\text{conf}_0[s](0, 1, \sqcup, 0^{2^n}, 1^n, 1, z_2, z_1, z_3, z_4)$. Such an atom, which is associated with the configuration $z_2$, contains all the auxiliary constants that will allow us to generate, via an additional set of DTGDs, all the $2^n$ atoms of the form $\text{conf}[s](b_1, \ldots, b_n, a, h, z_2, z_1, z_3, z_4)$ that perfectly describe the configuration $z_2$. The above informal description is illustrated in Figure 5.

— The trees labeled by atoms of the form $\text{conf}_0[s](\ldots)$ are generated by the following DTGDs: for each $s \in S_4$,
Fig. 5: Computation tree.

conf_0[s](Z, O, B, Z^{2n}, O^n, H, T, P, N_1, N_2) \rightarrow \\
\bigvee_{(s_1, s_2) \in S_3 \times S_3} \exists N_3 \exists N_4 \exists N_5 \exists N_6 \text{conf}_0[s_1](Z, O, B, Z^{2n}, O^n, H, N_1, T, N_3, N_4), \text{conf}_0[s_2](Z, O, B, Z^{2n}, O^n, H, N_2, T, N_5, N_6),

and for each \( s \in S_3, \)

\text{conf}_0[s](Z, O, B, Z^{2n}, O^n, H, T, P, N_1, N_2) \rightarrow \\
\bigvee_{s' \in S_\nu} \exists N_3 \exists N_4 \text{conf}_0[s'](Z, O, B, Z^{2n}, O^n, H, N_1, T, N_3, N_4) \lor \\
\exists N_3 \exists N_4 \text{conf}_0[s'](Z, O, B, Z^{2n}, O^n, H, N_2, T, N_3, N_4).

— The \( 2^n \) atoms, which perfectly describe a certain configuration, are generated as follows. First, we employ a stratified set of DIDs, consisting of \( n \) levels, in order to generate all the necessary tuples, and store them in predicates of the form \( \text{conf}_n[s] \). In particular, for each \( i \in \{0, \ldots, n-1\} \), and for each \( s \in S \):

\text{conf}_i[s](Z, O, B, Z^{2n-2i}, O^{n-i}, X^i, H, T, P, N_1, N_2) \rightarrow \varphi_1 \lor \varphi_2,

where

\[ \varphi_1 = \text{conf}_{i+1}[s](Z, O, B, Z^{2n-2i-2}, O^{n-i-1}, X^i, Z_{2n-2i-1}, H, T, P, N_1, N_2), \]

\[ \text{conf}_{i+1}[s](Z, O, B, Z^{2n-2i-2}, O^{n-i-1}, X^i, O_{n-i}, Z_{2n-2i-1}, T, P, N_1, N_2) \]
and
\[
\varphi_2 = \text{conf}_{i+1}[s](Z, O, B, Z^{2n-2i-2}, O^{n-i-1}, X^i, Z_{2n-2i-1}, Z_{2n-2i}, T, P, N_1, N_2),
\]
\[
\text{conf}_{i+1}[s](Z, O, B, Z^{2n-2i-2}, O^{n-i-1}, X^i, O_{n-i}, H, T, P, N_1, N_2).
\]

Now, having the relation \(\text{conf}_n[s]\), for each \(s \in S\), in place, we are ready to generate the completed \(\text{conf}[s]\) atoms by guessing the cell content of each cell. This can be done via the following DTGDs: for each \(s \in S\),
\[
\text{conf}_n[s](Z, O, B, X^n, H, T, P, N_1, N_2) \rightarrow \bigvee_{a \in \Lambda} \text{conf}[s](X^n, f(a), H, T, P, N_1, N_2),
\]
where \(f(0) = Z, f(1) = O\) and \(f(\top) = B\).

The construction of \(\Sigma\) is now completed. It is easy to verify that \(N(\Sigma)\) is a set of DIDs. This is because, for each \(\sigma \in \Sigma\), \(\sigma\) is constant-free, \(|\text{body}(\sigma)| = 1\), and there are no repeated variables in the atoms occurring in \(\sigma\).

The AUCQ \(Q\). It remains to check, via the AUCQ \(Q\), that each model of \(D \cup \Sigma\) encodes a tree that represents an invalid computation tree of \(M\). Roughly speaking, \(Q\) consists of the following disjuncts:

1. \(Q_{\text{initial}}\) for checking that in the initial configuration the cursor is not at the first cell, and the tape is not empty;
2. \(Q_{\text{trans}}\) for checking that the transition function of \(M\) is violated;
3. \(Q_{\text{move}}\) for detecting a wrong move of the cursor during a transition, i.e., a move which is neither to the left nor to the right; and
4. \(Q_{\text{inertia}}\) for checking that the tape cells not under the cursor do not keep their old values during a transition.

Let us now formalize the components of \(Q\).

1. \(Q_{\text{initial}}\) is defined as (recall that \(c \in C\) represents the initial configuration)
\[
\exists C \forall P \exists N_1 \exists N_2 \left( \text{conf}[s_0](0^n, C, 0, c, P, N_1, N_2) \right) \lor \\
\bigvee_{a \in \Lambda \setminus \{\top\}} \exists X^n \exists H \forall P \exists N_1 \exists N_2 \left( \text{conf}[s_0](X^n, a, H, c, P, N_1, N_2) \right).
\]

Notice that the first disjunct encodes the fact the the cursor is not at the first cell, and this is sufficient since, by construction, exactly one cell is pointed by the cursor.

2. With \(Q_{\text{trans}}\) we need to check that during a transition of \(M\) the state and the cell content has changed appropriately, and the head has moved to the correct position. Note that two subsequent cell positions agree on some initial portion of their binary encoding, which for the preceding cell is then followed by a 0 and then all 1s, and the other way around for the succeeding cell. We can thus connect the cell under the head in one configuration and the same cell in the next configuration, plus the appropriate adjacent cell in the next configuration. In the definition of \(Q_{\text{trans}}\), we make use of the functions \(f, g : \{-1, +1\} \rightarrow \{0, 1\}\), where \(f(-1) = g(1) = 1\) and \(f(+1) = g(-1) = 0\). Also, we make use of the binary operator \(\oplus^s\), where \(s \in S\), which is defined as \(\lor\), if \(s \in S_{\overline{3}}\), and as \(\land\), if \(s \in S_0\). Recall that \(\delta_\lor\) and \(\delta_\land\) denote the complement of \(\delta\) relative to universal and existential, respectively; for details, see
the proof of Theorem 4.5. \( Q_{\text{trans}} \) is defined as
\[
\bigvee_{s} \bigcap_{(s,a) \in \{(s_1, a_1, d_1), (s_2, a_2, d_2)\} \in \{1,2\}} \bigvee_{0 \leq j \leq n} \exists X^{n-j} E \in T \exists P \in N_1 \exists N_2 \exists P' \exists \exists N'_1 \exists N'_2 \exists C
\]
\[
\bigvee_{0 \leq i \leq n} \bigvee_{(s,a') \in S \times S} \bigwedge_{(a,a') \in \Lambda \times \Lambda, a \neq a'} \bigwedge_{i \in \{1,2\}} \bigwedge_{0 \leq j \leq n} \exists X^{n-j} E \in T \exists P \in N_1 \exists N_2 \exists P' \exists \exists N'_1 \exists N'_2 \exists C
\]

(3) The idea underlying \( Q_{\text{move}} \), is to check that in a certain configuration the cursor points the \( k \)-th cell, while in the previous configuration the cursor is not at the \((k-1)\)-th cell. The formal definition follows:
\[
\bigvee_{0 \leq i \leq n} \bigvee_{(s,a') \in S \times S} \bigwedge_{(a,a') \in \Lambda \times \Lambda, a \neq a'} \bigwedge_{i \in \{1,2\}} \bigwedge_{0 \leq j \leq n} \exists X^{n-j} E \in T \exists P \in N_1 \exists N_2 \exists P' \exists \exists N'_1 \exists N'_2
\]

Intuitively, the first and the third atom of a disjunct in the above query access the cell \( k \), while the second and the fourth atom access the cell \((k+1)\) and \((k-1)\), respectively. Notice that, although we do not force \( (X^{n-i}, 0, 1^{i-1}) \) and \( (Y^{n-j}, 1, 0^{j-1}) \) to have the same image during the evaluation of the query (otherwise, the above query will be cyclic), this is guaranteed by the fact that only one cell is pointed by the cursor (this holds by construction).

(4) Finally, \( Q_{\text{inertia}} \) is defined as
\[
\bigvee_{(s,a') \in S \times S} \bigwedge_{(a,a') \in \Lambda \times \Lambda, a \neq a'} \bigwedge_{i \in \{1,2\}} \bigwedge_{0 \leq j \leq n} \exists X^{n-j} E \in T \exists P \in N_1 \exists N_2 \exists P' \exists \exists N'_1 \exists N'_2
\]

This completes the construction of \( Q \). It is not difficult to verify that every disjunct of \( Q \) is an ACQ, and thus \( Q \) is acyclic.

By construction, and the assumption that a rejecting configuration does not have a successor configuration, while an accepting configuration has only itself as a successor, it is not difficult to see that \( D \cup \Sigma = Q \) iff \( M \) rejects \( I \) (see the argument given in the proof of Theorem 5.1, which shows that the employed construction is correct).

We now focus on frontier-guarded DTGDs, and show that query answering remains 2ExpTime-hard, even if we consider acyclic queries and predicates of bounded arity.

**Theorem 6.5.** ACQ answering under frontier-guarded DTGDs is 2ExpTime-hard, even for predicates of bounded arity.

**Proof.** The claim can be easily established by exploiting the fact that a CQ can be conceived as a frontier-guarded TGD, and give a reduction from CQ answering under frontier-guarded DTGDs which is 2ExpTime-hard, even for predicates of bounded arity. Consider a database \( D \), a set \( \Sigma \) of DTGDs, where the arity of the predicates of \( sch(\Sigma) \) is bounded by an integer constant, and a CQ \( q = \exists X \varphi(X) \). It is straightforward
### Table III: Complexity of answering atomic queries under guarded-based DTGDs.

<table>
<thead>
<tr>
<th>Class</th>
<th>Combined Complexity</th>
<th>Bounded Arity</th>
<th>Data Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DID</td>
<td>EXPTIME</td>
<td>PTIME</td>
<td>in AC₀</td>
</tr>
<tr>
<td></td>
<td>LB: Thm. 7.14</td>
<td>LB: Thm. 7.16</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>EXPTIME</td>
<td>PTIME</td>
<td>in AC₀</td>
</tr>
<tr>
<td></td>
<td>UB: Thm. 7.10</td>
<td>UB: Thm. 7.11</td>
<td></td>
</tr>
<tr>
<td>ML</td>
<td>2EXPTIME</td>
<td>EXPTIME</td>
<td>coNP</td>
</tr>
<tr>
<td></td>
<td>LB: Thm. 7.15</td>
<td>LB: Thm. 7.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LB: [Alviano et al. 2012, Thm. 7]</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>2EXPTIME</td>
<td>EXPTIME</td>
<td>coNP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FG</td>
<td>2EXPTIME</td>
<td>2EXPTIME</td>
<td>coNP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LB: Thm. 6.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UB: Thm. 4.3</td>
<td></td>
</tr>
<tr>
<td>WG</td>
<td>2EXPTIME</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UB: Thm. 6.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LB: [Cali et al. 2013, Thm. 4.1]</td>
<td></td>
</tr>
<tr>
<td>WFG</td>
<td>2EXPTIME</td>
<td>2EXPTIME</td>
<td>EXPTIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UB: Thm. 4.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UB: Thm. 4.4</td>
<td></td>
</tr>
</tbody>
</table>

To see that \(D \cup \Sigma \models q\) iff \(D \cup \Sigma \cup \{\sigma_q\} \models q',\) where \(\sigma_q\) is the frontier-guarded TGD \(\varphi(X) \rightarrow p,\) with \(p\) be 0-ary predicate not occurring in \(\text{sch}(\Sigma),\) and \(q' = p.\) Since \(q'\) is trivially acyclic, the claim follows.

The last lower bound for A(U)CQ answering is the EXPTIME-hardness when we focus on fixed sets of DIDs. This is done by simulating the behavior of an alternating linear space Turing machine. The desired result follows since alternating LINSPACE already coincides with EXPTIME. For the formal proof see Appendix C.

**Theorem 6.6.** ACQ (resp., AUCQ) answering under fixed sets of DIDs is EXPTIME-hard, even for predicates of arity at most three (resp., two).

### 7. ATOMIC QUERIES

We continue our complexity analysis of query answering under the respective guarded-based classes of DTGDs by concentrating on atomic queries (CQ₁), i.e., CQs consisting of a single atom. As usual, we start by summarizing our results, and then proceed with our new upper and lower complexity bounds in Section 7.2 and 7.3, respectively.

#### 7.1. Overview

Let us first clarify that the setting where the given set of DTGDs is fixed coincides with the setting where both the set of DTGDs and the query are fixed (i.e., the data complexity). By fixing the set of DTGDs, we implicitly fix the arity of the underlying schema, and thus the size of the given atomic query. For this reason, we exclude from our complexity analysis the case where the set of DTGDs is fixed.

Table III summarizes the complexity of CQ₁ answering. Compared with the results presented in Table II, it is clear that for the formalisms that allow more than one body-atoms (i.e., all the considered classes excluding DIDs and linear DTGDs) the complexity of our problem does not change if we focus on atomic queries. However, for...
DIDs and linear DTGDs the complexity decreases significantly, and in fact we obtain our first tractability results. More precisely, the combined complexity decreases from $2\text{EXPTIME}$ to $\text{EXPTIME}$, in the case of predicates of bounded arity the complexity decreases from $\text{EXPTIME}$ to $\text{PTIME}$, and the data complexity decreases from coNP to $\text{AC}_0$. The novel results of this section follow:

**Upper Bounds:**

1. $\text{CQ}_1$ answering under linear DTGDs is in $\text{EXPTIME}$ in combined complexity (Theorem 7.10) — this is shown by exhibiting an alternating algorithm that, at each step of its computation, uses polynomial space;
2. $\text{CQ}_1$ answering under linear DTGDs is in $\text{PTIME}$ if we focus on predicates of bounded arity (Theorem 7.11) — the alternating algorithm mentioned above uses only logarithmic space when the arity is fixed; and
3. $\text{CQ}_1$ under linear DTGDs is in $\text{AC}_0$ in data complexity (Theorem 7.13) — by showing that our problem is first-order rewritable, i.e., it can be reduced to the problem of evaluating a first-order query (which depends only on the given set of DTGDs and the given query) over the input database.

**Lower Bounds:**

1. $\text{CQ}_1$ answering under DIDs is $\text{EXPTIME}$-hard in combined complexity (Theorem 7.14) — this is established by simulating the behavior of an alternating linear space Turing machine;
2. $\text{CQ}_1$ answering under multi-linear DTGDs is $2\text{EXPTIME}$-hard in combined complexity (Theorem 7.15) — by adapting the proof of Theorem 6.4 in such a way that each disjunct of the employed UCQ can be conceived as a multi-linear DTGD that can be added in the constructed set of DTGDs, and eventually answer a single propositional atom;
3. $\text{CQ}_1$ answering under DIDs is $\text{PTIME}$-hard, even for predicates of bounded arity (Theorem 7.16) — by simulating a logarithmic space Turing machine; and
4. $\text{CQ}_1$ answering under multi-linear DTGDs is $\text{EXPTIME}$-hard, even if we focus on predicates of bounded arity (Theorem 7.17) — by simulating an alternating polynomial space Turing machine; in fact, the proof of this result is an adaptation of the proof of Theorem 7.15 (which in turn is an adaptation of the one of Theorem 6.4) in such a way that the (polynomially many) cells are encoded as part of the predicate name, and thus the arity becomes fixed.

Several missing complexity bounds are obtained from results of Sections 4 and 6 (see Table III). Let us clarify that the $2\text{EXPTIME}$-hardness of $\text{CQ}_1$ answering under frontier-guarded DTGDs, in the case of predicates of bounded arity, is inherited from Theorem 6.5 since the latter holds even for queries consisting of a single propositional atom; this fact is implicit in the proof of Theorem 6.5. Finally, the following two results from the literature provide us with optimal lower bounds for the data complexity of our problem, and the entire picture of the complexity of $\text{CQ}_1$ answering is completed.

**Inherited Results:**

1. $\text{CQ}_1$ answering under multi-linear DTGDs is coNP-hard in data complexity [Alviano et al. 2012, Theorem 7]; and
2. $\text{CQ}_1$ answering under weakly-guarded DTGDs is $\text{EXPTIME}$-hard in data complexity [Cali et al. 2013, Theorem 4.1].

From the results of this section, we observe that atomic queries do not make the query answering problem easier under the classes of DTGDs that allow more than
atoms in the body of a DTGD. However, the complexity in the case of DIDs and linear DTGDs significantly decreases, and in fact we obtain the first (and, to the best of our knowledge, the only known) tractability results for query answering under classes of DTGDs. Let us now proceed with the formal proofs of our results.

7.2. Upper Bounds

Up to now, all the upper bounds presented in the previous sections have been established by reducing our problem to the problem of reasoning (either query answering or satisfiability) under expressive computation logics such as the guarded fragment of first-order logic. However, for CQ1, answering under linear DTGDs this is not possible. All the upper bounds that can be obtained by following the usual approach are not optimal, and thus more refined techniques are needed. Our plan of attack is to reduce CQ1 answering under linear DTGDs to the problem of deciding whether a proof-tree exists, that is, a tree structure that encodes the finite part of each model of the given database w.r.t. the given set of DTGDs due to which the query is entailed, and then exhibit an alternating algorithm for deciding whether such a structure exists.

Proof-Trees. Let us start by introducing the notion of the proof-tree of an atom w.r.t. a set of DTGDs, and establish some key properties. In what follows, we consider linear DTGDs in normal form (see Section 2) in order to simplify our technical definitions and proofs.

Definition 7.1 (Proof-tree). Consider an atom \( a \), where \( \text{dom}(a) \subseteq C \), and a set \( \Sigma \) of linear DTGDs in normal form. Let \( T \) be a labeled binary tree \((N,E,\lambda_1,\lambda_2)\), where the nodes of \( N \) are labeled by \( \lambda_1 \) with atoms that can be formed using predicates of \( \text{sch}(\Sigma) \) and terms of \((\text{dom}(a) \cup N)\), and the edges of \( E \) are labeled by \( \lambda_2 : E \rightarrow \Sigma \). We say that \( T \) is a proof-tree of \( a \) and \( \Sigma \) if:

- The root is labeled by \( a \);
- For each \( v \in N \), there exists \( \sigma \in \Sigma \) such that each edge of the form \((u,v) \in E\) is labeled by \( \sigma \), and the out-degree of \( v \) is \(|\text{head}(\sigma)|\);
- For each \( v \in N \), if \( v \) has a single outgoing edge \((v,u) \) labeled by \( p(X) \rightarrow \exists Y r(X, Y) \), then there is a homomorphism \( h \) such that \( h(p(X)) = \lambda_1(v) \), and there exists \( h' = h_{\lambda_2} \cup \{Y \rightarrow t \mid t \in N, t \notin \text{dom}(h(p(X)))\} \) such that \( \lambda_1(u) = h'(r(X, Y)) \); and
- For each \( v \in N \), assuming that the outgoing edges of \( v \) are labeled by a DTGD \( \sigma \) without an existentially quantified variable, there is a homomorphism \( h \) such that \( h(\text{body}(\sigma)) = \lambda_1(v) \), and \( \{h(\bar{b}) \mid \bar{b} \in \text{head}(\sigma)\} = \{\lambda_1(u) \mid (v,u) \in E\} \).

A proof-tree \( T \) of \( a \) and \( \Sigma \) is valid w.r.t. to a CQ1 of the form \( \exists X p(X) \) if, for each leaf \( u \) of \( T \), there exists a homomorphism \( h \) such that \( \lambda_1(u) = h(p(X)) \).

Example 7.2. Consider the set \( \Sigma \) of linear DTGDs consisting of

\[
\sigma_1 : p(X,Y) \rightarrow r(Y,X) \\
\sigma_2 : r(X,Y) \rightarrow p(X,Y) \lor t(X,X) \\
\sigma_3 : t(X,Y) \rightarrow p(X,Y) \\
\sigma_4 : r(X,Y) \rightarrow \exists Z s(Y,Z).
\]

Possible proof-trees of the atom \( r(a,b) \) and \( \Sigma \) are shown in Figure 6. In particular, tree (a) is valid w.r.t. the atomic queries \( \exists X s(a, X) \) and \( \exists X \exists Y s(X,Y) \), while tree (b) is valid w.r.t. \( \exists X s(b, X) \) and \( \exists X \exists Y s(X,Y) \).

The following key lemma, established in [Alviano et al. 2012, Lemma 4], shows that for CQ1 answering under linear DTGDs one can focus on a single database atom; for the proof see Appendix D:
Fig. 6: Possible proof-trees of $r(a, b)$ and $\Sigma$.

**Lemma 7.3.** Consider a database $D$, a set $\Sigma$ of linear DTGDs in normal form, and a CQ$_1$ $q$. Then, $D \cup \Sigma \models q$ iff there exists $a \in D$ such that $\{a\} \cup \Sigma \models q$.

Although we can focus on a single database atom, in general we have to consider infinitely many models. The key idea underlying our approach is to use "representatives" of all these models in order to be able to encode them in a single structure, namely the proof-tree defined above. This can be achieved by considering the skolemized version of the given set of DTGDs. Given a set $\Sigma$ of linear DTGDs, we define $F_\Sigma$ as the set of skolem functions $\{f_\sigma \mid \sigma \in \Sigma\}$, where the arity of each $f_\sigma$ is the number of universally quantified variables of $\sigma$. Let $\Sigma_f$ denote the set of rules obtained from $\Sigma$ by replacing each TGD $\sigma$ of the form $p(X) \rightarrow \exists Y r(X, Y)$ with the rule $p(X) \rightarrow r(X, f_\sigma(X))$. From well-known results on skolemization the following holds: Given a database $D$, a set $\Sigma$ of linear DTGDs in normal form, and a CQ$_1$ $q$, $D \cup \Sigma \models q$ iff $D \cup \Sigma_f \models q$. The set of skolem terms $\Gamma_\Sigma$ is recursively defined as follows: each term of $\mathcal{C}$ belongs to $\Gamma_\Sigma$, and if $f_\sigma \in F_\Sigma$ has arity $n > 0$ and $t_1, \ldots, t_n$ are terms of $\Gamma_\Sigma$, then $f_\sigma(t_1, \ldots, t_n) \in \Gamma_\Sigma$. The notion of homomorphism naturally extends to atoms that contain functional terms of the form $f(t_1, \ldots, t_n)$, where each $t_i$ is a variable of $\mathcal{V}$ or a skolem term; in particular, given a homomorphism $h$, $h(f(t_1, \ldots, t_n)) = f(h(t_1), \ldots, h(t_n))$. In what follows, it is more convenient to conceive models as trees. In fact, as we shall see, this will allow us to easily build the desired proof-tree of an atom and a set of DTGDs.

**Definition 7.4** ($\rho$-tree). Consider a set $\Sigma$ of linear DTGDs in normal form, an atom $\mathfrak{a}$, where $dom(\mathfrak{a}) \subset \Gamma_\Sigma$, and an instance $M \in mods(\{\mathfrak{a}\}, \Sigma_f)$. The tree of $M$, denoted $tree(M)$, is a labeled rooted tree $(N, E, \lambda_1, \lambda_2)$, where $N$ is the node set, $E$ is the edge set, $\lambda_1 : N \rightarrow M$ is a node-labeling function, and $\lambda_2 : E \rightarrow \Sigma_f$ is an edge-labeling function, such that:

---

The root of $tree(M)$ is labeled by $\mathfrak{a}$; and

For each node $v \in N$, and rule $\rho \in \Sigma_f$ for which there exists a homomorphism $h$ that maps body($\rho$) into $\lambda_1(v)$, the following holds: for each $\hat{e} \in head(\rho)$, if $h(\hat{e}) \in M$, then there exists $u \in N$ with $\lambda_1(u) = h(\hat{e})$, $e = (v, u)$ belongs to $E$, and $\lambda_2(e) = \rho$.

---

The $\rho$-tree of $M$, denoted $\rho$-tree($M$), is the tree obtained from $tree(M)$ by keeping the root node $\mathfrak{a}$, each edge $e = (v, u)$ which is labeled by $\rho$, and the subtree rooted at $u$. Let $(\rho)$-trees($\mathfrak{a}, \Sigma_f$) = $\{\rho$-tree($M$) | $M \in mods(\{\mathfrak{a}\}, \Sigma_f)\}$. By abuse of notation, given a ($\rho$)-tree $T$ and a CQ$_1$ $q$, we write $T \models q$ if $\{\lambda_1(v)\}_{v \in N} \models q$. $\square$
Consider the set

\[ p(\mathbf{i}) \]

whether a proof-tree exists. The latter can be tackled via an alternating algorithm.

A CQ of the next crucial result, which can be found in Appendix D.

Guarded-Based Disjunctive Tuple-Generating Dependencies A:37

\[ \text{Fig. 7: Possible trees of } \text{trees}(d, \Sigma_f). \]

**Example 7.5.** Consider the set \( \Sigma_f \) consisting of

\[ \begin{align*}
\rho_1 : & \quad p(X, Y) \rightarrow r(Y, X) \\
\rho_2 : & \quad r(X, Y) \rightarrow p(X, Y) \lor t(X, X) \\
\rho_3 : & \quad t(X, Y) \rightarrow p(X, Y) \\
\rho_4 : & \quad r(X, Y) \rightarrow s(Y, f(X, Y)).
\end{align*} \]

Possible models of \( \{r(a, b)\} \cup \Sigma_f \) are:

\[ \begin{align*}
M_1 &= \{r(a, b), s(b, f(a, b)), t(a, a), p(a, a), r(a, a), s(a, f(a, a))\}, \\
M_2 &= \{r(a, b), s(b, f(a, b)), p(a, b), r(b, a), t(b, b), p(b, b), r(b, b), s(a, f(b, a)), s(b, f(b, b))\}, \\
M_3 &= \{r(a, b), s(b, f(a, b)), p(a, b), r(b, a), p(b, a), s(a, f(b, a))\}.
\end{align*} \]

Notice that, for each other model \( M \) of \( \{r(a, b)\} \cup \Sigma_f \), it holds that \( M_i \subseteq M \), for at least one \( i \in [3] \); the tree \( T_i \) of \( M_i \) is depicted in Figure 7. Observe that each \( T_i \) has exactly one \( \rho_2 \)-tree and one \( \rho_1 \)-tree. Moreover, the shaded paths form (modulo null renaming) the proof-tree of \( r(a, b) \) and \( \Sigma_f \) shown in Figure 6(a), which is valid w.r.t. \( s(a, X) \).

Observe that the first edge of each shaded path in Figure 7 is labeled by the same rule. As shown in the next technical lemma, this is a general property that holds whenever the given database and set of DTGDs entail the given atomic query.

**Lemma 7.6.** Consider a set \( \Sigma \) of linear DTGDs in normal form, an atom \( a \), where \( \text{dom}(a) \subset C \), and a CQ \( q \). It holds that, \( \{a\} \cup \Sigma_f \models q \) iff there exists \( \rho \in \Sigma_f \) such that \( T \models q \), for each \( T \in \rho\text{-trees}(a, \Sigma_f) \).

Given an atom \( a \), where \( \text{dom}(a) \subset C \), a set \( \Sigma \) of linear DTGDs, and a CQ \( q \), by applying recursively the property guaranteed by Lemma 7.6, one can build a proof-tree of \( a \) and \( \Sigma \) which is valid w.r.t. \( q \). This is exactly the key idea underlying the proof of the next crucial result, which can be found in Appendix D.

**Lemma 7.7.** Consider a database \( D \), a set \( \Sigma \) of linear DTGDs in normal form, and a CQ \( q \). It holds that, \( D \cup \Sigma \models q \) iff there exists a proof-tree of \( a \) and \( \Sigma \), for some \( a \in D \), which is valid w.r.t. \( q \).

By Lemma 7.7 we get that CQ\( \_3 \) answering is equivalent to the problem of deciding whether a proof-tree exists. The latter can be tackled via an alternating algorithm.
Consider a database \( N \) and \( N(\Sigma) \), for some \( a \in D \), which is valid w.r.t. \( q \).

(1) \( \Sigma := N(\Sigma) \);

(2) \( N := \{ z_i \in N \}_{i \in [m+1]} \), where \( m \) is the maximum arity over all predicates of \( \text{sch}(\Sigma) \);

(3) Guess an atom \( a \in D \);

(4) Guess to either execute step 5 or to skip to step 6;

(5) If there exists a homomorphism \( h \) such that \( h(p(X)) = a \), then accept;

(6) Guess a DTGD \( \sigma \in \Sigma \), the set \( \text{sch}(\Sigma) \) of the form \( r(X) \to \exists Y \psi(X,Y) \) (\( Y \) may not be present), and a homomorphism \( h \) such that \( h(\text{body}(\sigma)) = a \); if there is no such \( \sigma \) and \( h \), then reject;

(7) Universally choose each disjunct \( b \) of \( \text{head}(\sigma) \) and do the following:

(a) If \( Y \) is present in \( \sigma \), then \( h' := h |_X \cup \{ Y \to t \mid t \in N \land t \not\in \text{dom}(h(r(X))) \} \);

otherwise, \( h' := h \);

(b) \( a := h'(b) \) and goto step 4.

Fig. 8: The alternating algorithm SearchPT.

The Alternating Algorithm. The alternating algorithm called SearchPT takes as input a database \( D \), a set \( \Sigma \) of linear DTGDs, and a CQ \( q \), and decides whether a proof-tree of \( a \) and \( N(\Sigma) \), for some \( a \in D \), which is valid w.r.t. \( q \) exists. The formal definition of SearchPT is depicted in Figure 8, while an example of its computation follows.

Example 7.8. Consider the database \( D = \{ r(a,b) \} \), the set \( \Sigma \) of linear DTGDs

\[
\begin{align*}
\sigma_1 : r(X,Y) &\to p(X) \lor t(Y,X) \\
\sigma_2 : p(X) &\to r(X,X) \\
\sigma_3 : t(X,Y) &\to \exists Z s(Y,Z) \\
\sigma_4 : s(X,Y) &\to \exists Z s(Y,Z),
\end{align*}
\]

and the CQ \( q = \exists X p(X) \). Figure 9 shows an initial part of the alternating computation of SearchPT\((D,\Sigma,q)\). Observe that in the shaded edge exactly three (i.e., maximum arity plus one) nulls appear.

Correctness of SearchPT follows by construction. Lemma 7.7 implies the following:

**Proposition 7.9.** Consider a database \( D \), a set \( \Sigma \) of linear DTGDs, and a CQ \( q \). Then, \( D \cup \Sigma \models q \) iff SearchPT\((D,\Sigma,q)\) accepts.

**Complexity Upper Bounds.** Equipped with the above machinery, we are now ready to establish the desired complexity upper bounds for our problem.

**Theorem 7.10.** CQ\(_1\) answering under linear DTGDs is in \( \text{ExpTIME} \) in combined complexity.

**Proof.** Consider a database \( D \), a set \( \Sigma \) of linear DTGDs, and a CQ \( q \). Recall that alternating PSPACE coincides with EXPTIME. Thus, by Proposition 7.9, and since SearchPT is an alternating procedure, it suffices to show that SearchPT\((D,\Sigma,q)\) uses polynomial space in general. Clearly, the set of nulls \( N \) can be maintained in \( O(m \log m) \) space, where \( m \) is the maximum arity over all predicates of \( \text{sch}(N(\Sigma)) \). Furthermore, the atom \( h'(b) \) can be maintained in \( O(m \log m + m \log n) \) space, where \( n = |\text{dom}(D)| \).

This is because \( h'(b) \) contains at most \( m \) terms, and each term can be represented using \( \log m \) bits, if is a null of \( N \), or \( \log n \) bits, if is a constant of \( \text{dom}(D) \). Therefore, at each step of its computation the algorithm SearchPT uses \( O(m \log m + m \log n) \) space. \( \square \)
We proceed to show that CQ₁ answering under linear DTGDs is in PTIME in case of predicates of bounded arity. Since alternating LOGSPACE coincides with PTIME, it suffices to show that SearchPT(D, Σ, q) uses only logarithmic space. Recall that our algorithm uses O(m log m + m log n) space, where m is the maximum arity over all predicates of sch(N(Σ)). However, the existing normalization procedure (see Section 2) introduces new auxiliary predicates of unbounded arity, and thus m is not an integer constant. Thus, we need a normalization procedure that does not increase the arity of the original schema. Although it is not clear how such a normalization procedure works in general, for CQ₁ answering under linear DTGDs such a procedure can be easily defined. The key fact is that we do not need to preserve the joins among nulls since linear DTGDs can have only one body-atom, and also the query to be answered is a single atom. The normalization procedure can be found in Appendix D. With such a normalization procedure in place we get the following:

**Theorem 7.11.** CQ₁ answering under linear DTGDs is in PTIME in case of predicates of bounded arity.

The rest of this subsection is devoted to show that CQ₁ answering under linear DTGDs is in AC₀ in data complexity. We do this by establishing that our problem is first-order rewritable, i.e., it can be reduced to the problem of evaluating a first-order query over a database. First-order rewritability was first introduced in the context of description logics [Calvanese et al. 2007].

Consider a set Σ of linear DTGDs, and a CQ₁ q. Let C be the constants occurring in q, and N = {z₁, …, zm} be a set of nulls, where m is the maximum arity over all predicates of sch(Σ). Let base(q, Σ) be the set of all atoms that can be formed using terms of C ∪ N and predicates of sch(Σ). Let B = {b | b ∈ base(q, Σ) and {b} ⊨ Σ ⊨ q}, and µ be a renaming substitution that maps each z ∈ N into a distinct variable Xz ∈ V. We define the first-order query qΣ as

\[ \bigvee_{b \in B} \exists X_{z_1} \ldots \exists X_{z_m} \mu(b). \]

In fact, qΣ is a union of CQs, where each disjunct is an atomic CQ. It is easy to see that |B| ≤ |sch(Σ)| · (2m)m. Since, by Theorem 7.10, CQ₁ answering under linear DTGDs is
feasible in exponential time, we conclude that the set \( B \), and thus the query \( q_\Sigma \), can be constructed in exponential time. In what follows, we show that \( q_\Sigma \) is a sound and complete rewriting:

**Lemma 7.12.** Consider a set \( \Sigma \) of linear DTGDs in normal form, and a CQ\(_1\) \( q \). Then, for every database \( D \), \( D \cup \Sigma \models q \) iff \( D \models q_\Sigma \).

**Proof.** Fix a database \( D \) and let \( q = \exists X p(X) \). By construction of \( q_\Sigma \), it suffices to show that \( D \cup \Sigma \models q \) iff there exists \( b \in \text{base}(q, \Sigma) \) and a homomorphism \( h \) such that \( h(b) \in D \) and \( \{b\} \cup \Sigma \models q \).

\((\Rightarrow)\) By Lemma 7.3, there exists an atom \( a \in D \) such that \( \{a\} \cup \Sigma \models q \). Therefore, for each \( M \in \text{mods}(\{a\}, \Sigma) \), there exists a homomorphism \( h_M \) such that \( h_M(p(X)) \subseteq M \).

Moreover, by construction, there exists \( b \in \text{base}(q, \Sigma) \) and a bijective homomorphism \( \lambda \) such that \( \lambda(b) = a' \); clearly, \( \lambda(b) \in D \). Let \( \gamma : \text{dom}(a) \rightarrow \text{dom}(b) \) be the substitution \( \{t \rightarrow t' \mid t \rightarrow t \in \lambda^{-1}(\text{dom}(\Sigma')) \} \). By induction on the depth of the trees of the models, it can be shown that for each \( M' \in \text{mods}(\{b\}, \Sigma) \), there exists \( M \in \text{mods}(\{a\}, \Sigma) \) such that \( \gamma(M) = M' \); thus, the homomorphism \( \gamma \circ h_M \) maps \( p(X) \) into \( M' \).

\((\Leftarrow)\) This direction can be shown by providing a similar argument. \( \square \)

In the case of data complexity, the set of DTGDs \( \Sigma \) and the CQ\(_1\) \( q \) are fixed, and thus \( q_\Sigma \) is fixed. Since the evaluation of first-order queries is feasible in AC\(_0\) when the query is fixed [Vardi 1995], Lemma 7.12 implies the following complexity result:

**Theorem 7.13.** CQ\(_1\) answering under linear DTGDs is in AC\(_0\) in data complexity.

This concludes the upper bound section for atomic queries.

### 7.3. Lower Bounds

We proceed now to establish the desired complexity lower bounds for CQ\(_1\) answering under DIDs. The following theorem is obtained by simulating an alternating polynomial space Turing machine, using the ideas of the PS\(_{1}\)-hardness proof of the implication problem of (non-disjunctive) IDs given in [Casanova et al. 1984].

**Theorem 7.14.** CQ\(_1\) answering under DIDs is EXPTIME-hard in combined complexity.

**Proof.** The proof is by reduction from the acceptance problem of an alternating linear space Turing machine \( M \) on input \( I = a_1 \ldots a_{|I|} \). Let \( M = (S, \Lambda, \delta, s_0) \), where \( S = S_\emptyset \sqcup S_\exists \sqcup \{s_a\} \sqcup \{s_r\} \) is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, \( \Lambda = \{0, 1, \sqcup\} \) is the tape alphabet with \( \sqcup \) the blank symbol, \( \delta : S \times \Lambda \rightarrow (S \times \Lambda) \) is the transition function, and \( s_0 \in S \) is the initial state. We assume that \( M \) is well-behaved and never tries to read beyond its tape boundaries, always halts, and uses exactly \( n = |I| \) tape cells. Also, for each transition \( (s, a) \rightarrow ((s', a', d), (s'', a'', d'')) \), we assume that \( d' = d'' \); the latter is a technical assumption which will allow us to represent the transitions of \( \delta \) in a more convenient form (in the same way as in the PSPACE-hardness proof of the implication problem of (non-disjunctive) IDs given in [Casanova et al. 1984]). We represent configurations using a string \( \Lambda^* S \Lambda^* \), i.e., the state of the configuration is placed to the immediate left of the cursor position. In this notation, the initial configuration is \( s_0 a_1 \ldots a_{|I|} \). Finally, we assume that the accepting configuration of \( M \) is \( s_\emptyset \sqcup \sqcup^n \).

Our goal is to construct a database \( D \), a set \( \Sigma \) of DIDs, and a CQ\(_1\) \( q \) such that \( D \cup \Sigma \models q \) iff \( M \) accepts. The idea is to represent the configurations of \( M \) by strings of length \( (|S| + |\Lambda|) \cdot (n+1) \), which are stored in the predicate \( \text{conf} \). Each symbol of \( (S \cup \Lambda) \) will be represented by a string of length \( (|S| + |\Lambda|) \) of the form \( \{|0\} \cup N \}^* \).
To this end, we assume the order $s_0 < s_1 < \ldots < s_{|S|-1} < 0 < 1 < \sqcup$ on the elements of $(S \cup \Lambda)$. A symbol $x \in (S \cup \Lambda)$ that occurs at the $i$-th position of the above order is represented by a string $\{0\}^{i-1} \{1\}^{|S|+|\Lambda|-i+1}$. For example, a possible encoding for $s_0$ is $1, 0, \ldots, 0$, while for $\sqcup$ is $0, \ldots, 1$. Now, given a configuration $C = x_1, x_2, \ldots, x_{n+1}$, the encoding of $C$ is a string of length $(|S| + |\Lambda|) \cdot (n + 1)$, where at its $i$-th position $a$ possible encoding of $x_i$ occurs. Formally, assuming a total order on the elements of $(S \cup \Lambda) \times [n+1]$, where $(x, i) < (y, j)$ iff $(i < j)$ or $(i = j$ and $x < y)$, with $\#(x,i)$ be the position of $(x, i)$ in this order, the encoding of a configuration $C$ contains 1 at position $\#(x,i)$ iff the $i$-th symbol in $C$ is $x$. We proceed with the construction of $D, \Sigma$ and $q$.

The Database $D$. $D$ consists of a single atom which represents the initial configuration $s_0 a_1 \ldots a_{|\ell|}$ of $M$. Given two elements $x, y \in \Lambda$, $f(x, y) = 1$ if $x = y$; otherwise, $f(x, y) = 0$. The database atom is defined as $\{\text{conf}(t)\}$, where $t$ is the binary tuple:

$$t = \begin{cases} 0, & [0, \ldots, 0], \quad 1, & [1, \ldots, 1], \\ 0, & [0, \ldots, 0], \quad f(a_1, 0), \quad f(a_1, 1), \quad f(a_1, \sqcup), \\ 0, & [0, \ldots, 0], \quad f(a_{|\ell|}, 0), \quad f(a_{|\ell|}, 1), \quad f(a_{|\ell|}, \sqcup). \end{cases}$$

Clearly, $t$ is obtained by replacing each symbol occurring in the initial configuration of $M$, by its (unique) encoding that contains only symbols of $\{0, 1\}$.

The Set $\Sigma$. With $\Sigma$ we encode the transitions of $\delta$. These transitions can be described by expressions of the form $x_1 y_1 z_1 \rightarrow x_2 y_2 z_2; x_3 y_3 z_3$ with $x_1, y_1, z_1 \in (S \cup \Lambda)$, for each $i \in \{1, 2, 3\}$. For example, the transition $(s, a) \rightarrow ((s', a', -1), (s'', a'', -1))$ corresponds to $xsa \rightarrow s'xa'; s''xa''$, for each $x \in \Lambda$, while the transition $(s, a) \rightarrow ((s', a', +1), (s'', a'', +1))$ corresponds to $sax \rightarrow a's'x; a''s''x$, for each $x \in \Lambda$. Let $T_3$ be all such expressions corresponding to transitions of $\delta$.

Consider an expression $\tau \in T_3$ of the form $x_1 y_1 z_1 \rightarrow x_2 y_2 z_2; x_3 y_3 z_3$. For each $i \in \{1, 2, 3\}$ and $j \in [n-1]$, we define the atom $g^i_\tau$ as $\text{conf}(t)$, where $t \in V^{ar}$ is as follows:

- At positions $\#(x, i)$, $\#(y, i+1)$ and $\#(z, i+2)$ of $t$ the variables $X$, $Y$ and $Z$ occur, respectively;
- For each $(x, \ell) \in (\Lambda \times \{1, \ldots, j-1, j+3, \ldots, n+1\})$, at position $\#(x, \ell)$ of $t$ the variable $X_\ell$ occurs; and
- At each position of $t$ not considered above a distinct variable occurs; these variables form the set $Y_\ell$.

Having the above atoms in place, we are now ready to define the set $\Sigma$ of DIDs. For each $\tau = x_1 y_1 z_1 \rightarrow x_2 y_2 z_2; x_3 y_3 z_3$ of $T_3$, where $s \in S$ is the state occurring in $x_1 y_1 z_1$, and for each $j \in [n-1]$:  

- If $s \in S_Y$, then $\Sigma$ includes the DID
  $$g^1_\tau \rightarrow \exists Y_2 g^2_\tau \lor \exists Y_3 g^3_\tau.$$

- If $s \in S_S$, then $\Sigma$ includes the two (non-disjunctive) IDs
  $$g^1_\tau \rightarrow \exists Y_2 g^2_\tau \quad \text{and} \quad g^1_\tau \rightarrow \exists Y_3 g^3_\tau.$$

This completes the construction of $\Sigma$.

The CQ $q$. The query $q$ is defined as $\exists X \text{conf}(t)$, where $t \in (\{1\} \cup X)^{ar}$ as is as follows:

- At positions $\#(s_{a, 1}), \#(\sqcup, 2), \ldots, \#(\sqcup, n+1)$ the constant 1 occurs; and
- At all the other positions a distinct variable of $X$ occurs.
The construction of \( q \) is now completed.

By construction, whenever a transition \( \tau \in \delta \) would take place, at least one of the DIDs of \( \Sigma \) will propagate the constant 1 in such a way that the derived atom will represent the successor configuration. If the machine thus reaches the accepting configuration, this implies that there exists a sequence of DIDs that will derive the atom corresponding to that accepting configuration, namely the CQ\(_1\) \( q \). Conversely, if \( D \cup \Sigma \models q \), then there exists a sequence of DIDs as above, and the sequence of corresponding transitions in \( \delta \) yields an accepting computation of \( M \). 

We now focus on multi-linear DTGDs, and show that query answering remains 2ExpTime-hard, even if we consider atomic queries. Let \( D, \Sigma \) and \( Q \) be the database, the set of DIDs, and the UCQ employed in the proof of Theorem 6.4, where it is shown that ACQ answering under DIDs is 2ExpTime-hard. We can construct in polynomial time a set \( \Sigma' \) of multi-linear DTGDs, which depends on \( \Sigma \) and \( Q \), such that \( D \cup \Sigma \models q \) iff \( D \cup \Sigma' \models q \), where \( q = p^* \) with \( p^* \) be an auxiliary 0-ary predicate not occurring in \( \text{sch}(\Sigma) \); for the formal construction see Appendix D. In fact, our intention is to obtain \( \Sigma' \) by adding to \( \Sigma \) the TGD \( \varphi \rightarrow p^* \), for each disjunct \( \varphi \) of \( Q \). The next result follows:

**Theorem 7.15.** CQ\(_1\) answering under multi-linear DTGDs is 2ExpTime-hard in combined complexity.

Although atomic query answering under DIDs is ExpTime-hard, in case of predicates of bounded arity we can show that is PTime-hard; in fact, this is true even if we focus on unary predicates. The proof, which can be found in Appendix D, is by reduction from the acceptance problem of an alternating logarithmic space Turing machine.

**Theorem 7.16.** CQ\(_1\) answering under DIDs is PTime-hard, even for unary predicates.

The last lower bound that we establish for CQ\(_1\) answering is the ExpTime-hardness when we focus on multi-linear DTGDs and predicates of bounded arity. This result can be obtained by adapting the proof of Theorem 7.15. Since we have to simulate an alternating polynomial space (and not exponential space) Turing machine, the polynomially many cells can be encoded in the predicates, and thus the arity becomes fixed. The proof can be found in Appendix D.

**Theorem 7.17.** CQ\(_1\) answering under multi-linear DTGDs is ExpTime-hard, even for predicates of bounded arity.

### 8. Querying Description Logic Knowledge Bases

Description logics (DLs) [Baader et al. 2003] have been playing a prominent role in ontological reasoning. Most DLs are decidable fragments of first-order logic, based on concepts (classes of objects) and roles (binary relations on concepts). Several variants of DLs have been proposed, where a central issue is the trade-off between the expressive power and the computational complexity of the reasoning services such as query answering. In this section, we show that our techniques and results on guarded DTGDs can be used as a generic tool for establishing results on query answering under several central DLs. To this end, we focus on the key DL \( \text{ALCH} \).

#### 8.1. The Description Logic \( \text{ALCH} \)

\( \text{ALC} \) has been introduced in [Schmidt-Schauß and Smolka 1991]. It is at the basis of many expressive DLs, and is the “smallest” DL that is propositionally closed, i.e., that
allows for all Boolean connectives. By enriching \( ALC \) with inverse roles (\( I \)) and role hierarchies (\( H \)) we obtain the expressive DL \( ALCITH \). Let \( N_C \) and \( N_R \) be disjoint countably infinite sets of concept and role names, respectively. The set of \( ALCITH \)-concepts is the smallest set such that: (1) every concept name \( A \in N_C \), as well as \( \top \) and \( \bot \), are \( ALCITH \)-concepts; and (2) if \( C \) and \( D \) are \( ALCITH \)-concepts and \( R \in N_R \), then \( \neg C, C \cap D, C \cup D, \exists R.C, \forall R^-C, \forall R.C \) and \( \forall R^-C \) are \( ALCITH \)-concepts. A DL knowledge base (KB) \( K = (T, A) \) represents the domain of interest in terms of two parts, a terminological box (TBox) \( T \), specifying the intensional knowledge, and an assertional box (ABox) \( A \), asserting the extensional knowledge. In an \( ALCITH \) KB the TBox is a finite set of concept inclusion assertions of the form \( B \subseteq C \), where \( B \) and \( C \) are \( ALCITH \)-concepts, as well as a finite set of role inclusion assertions of the form \( R \subseteq S \), where \( R \) and \( S \) are (possibly inverse) roles. The ABox consists of a finite set of assertions \( C(a) \) and \( R(a,b) \), where \( a \) and \( b \) are individuals of \( C \), \( C \) is an \( ALCITH \)-concept and \( R \) is a role name. The semantics of \( ALCITH \) is given by interpretations in the usual way; see Appendix E.

### 8.2. Generic Complexity Results

By exploiting our techniques and results, we can establish the following result about (BTW)UCQ answering: \( \mathcal{L}_1 \subseteq \mathcal{L}_2 \) means that \( \mathcal{L}_1 \) is a subformalism of \( \mathcal{L}_2 \):

**THEOREM 8.1.** Consider a DL \( \mathcal{L} \) such that \( \mathcal{L} \subseteq ALCITH \) and \( \mathcal{L} \) is powerful enough for expressing the following inclusion assertions:

\[
A_1 \subseteq A_2 \sqcup A_3 \quad A \subseteq \exists R.\top \quad \exists R^-\top \subseteq A \quad R \sqsubseteq S \quad R^- \sqsubseteq S,
\]

where \( A, A_1, A_2, A_3 \) are concept names and \( R, S \) are role names. Then, (BTW)UCQ answering under \( \mathcal{L} \) is 2\text{ExpTime}-complete in combined complexity, even when the TBox is fixed.

An analogous result for AUCQ answering can be also established:

**THEOREM 8.2.** Consider a DL \( \mathcal{L} \) such that \( \mathcal{L} \subseteq ALCITH \) and \( \mathcal{L} \) is powerful enough for expressing the following inclusion assertions:

\[
A_1 \subseteq A_2 \sqcup A_3 \quad A \subseteq \exists R.\top \quad \exists R^-\top \subseteq A,
\]

where \( A, A_1, A_2, A_3 \) are concept names. Then, AUCQ answering under \( \mathcal{L} \) is \text{ExpTime}-complete in combined complexity, even when the TBox is fixed.

The upper bounds for both theorems are obtained by reducing query answering under \( ALCITH \) to query answering under guarded DTGDs. The desired lower bounds are inherited from Theorems 4.6, 5.1 and 6.6. In fact, the sets of DIDs employed in the proofs of the above theorems can be rewritten as DL axioms; see Appendix E.

**A Case Study.** We conclude this section by briefly discussing which existing DLs benefit from our generic Theorems 8.1 and 8.2. Clearly, \( ALCITH \) itself is one of those DLs since it is powerful enough to express all the axioms listed in our theorems. Other DLs are \( DL-Lite_{boot}^H \) [Artale et al. 2009], one of the most expressive languages of the DL-Lite family, \( \mathcal{EL} \) [Baader et al. 2005], i.e., the extension of the well-known DL \( \mathcal{EL} \) with union of concepts, and \( \mathcal{ELU} \) and \( \mathcal{ELU} \), that is, the extension of \( \mathcal{ELU} \) with inverse roles and role hierarchies. The \( 2\text{ExpTime} \)-completeness of UCQ answering under \( ALCITH \) is actually known [Lutz 2008]. However, the \( \text{ExpTime} \)-completeness of AUCQ answering under \( ALCITH \) (even in the case of a fixed TBox), as well as all the inherited results for \( DL-Lite_{boot}^H, \mathcal{ELU} \) and \( \mathcal{ELU} \), are novel.

### 9. CONCLUSIONS

In this paper, we have studied in depth the complexity of (U)CQ answering under the main guarded-based classes of DTGDs that can be found in the literature. Interest-
ingly, the problem under consideration is 2EXPSPACE-hard even in the case of a fixed set of DIDs, that is, the simplest guarded-based class of DTGDs. With the aim of reducing the complexity of our problem, we considered three key subclasses of (U)CQs, namely (U)CQs of bounded hypertree-width, (U)CQs of bounded treewidth and acyclic (U)CQs. Our investigation shows that the above query languages do not have the expected positive impact on our problem. Finally, towards the identification of tractable cases, we considered atomic queries. We show that for DIDs and linear DTGDs the complexity significantly decreases, and we obtain the first tractability results for query answering under DTGD-based formalisms. Our techniques and results can be used as a generic tool for establishing complexity results for query answering under key DLs.

REFERENCES
Anonymous. I. Details are omitted due to double-blind reviewing.
Anonymous. II. Details are omitted due to double-blind reviewing.
Anonymous. III. Details are omitted due to double-blind reviewing.

ACM Transactions on Database Systems, Vol. V, No. N, Article A, Publication date: January YYYY.
Guarded-Based Disjunctive Tuple-Generating Dependencies


ACM Transactions on Database Systems, Vol. V, No. N, Article A, Publication date: January YYYY.
A. NORMAL FORM OF DTGDS

A set \( \Sigma \) of DTGDs is in normal form if each \( \sigma \in \Sigma \) is of the form

\[
\varphi(X) \rightarrow p(X) \quad \text{or} \quad \varphi(X) \rightarrow \exists Y p(X, Y) \quad \text{or} \quad \varphi(X) \rightarrow p_1(X) \lor p_2(X).
\]

Every set \( \Sigma \) of DTGDs can be transformed in logarithmic space into a set \( N(\Sigma) \) in normal form such that, for every database \( D \) and UCQ \( Q \), \( D \cup \Sigma \models Q \) iff \( D \cup N(\Sigma) \models Q \). The normalization procedure follows.

Consider a DTGD \( \sigma \). First, we construct the set \( N_1(\sigma) \) by exhaustively applying the following two replacement rules, starting from \( \sigma \), until each assertion is either a single-atom-head TGD, or a DTGD with a disjunction of two atoms in its head:

1. A DTGD of the form \( \varphi(X) \rightarrow \bigvee_{i=1}^n \exists Y_i \psi_i(X, Y_i) \) is replaced by
   \[
   \varphi(X) \rightarrow \exists Y_1 p_1^\star(X, Y_1) \lor \exists Y_2,..,Y_n p_2^\star(X, Y_2,..,Y_n)
   \]
   where \( Y_2,..,Y_n = \bigcup_{i=2}^n Y_i \), \( p_1^\star \) is an \(|X \cup Y_1|\)-ary predicate and \( p_2^\star \) is an \(|X \cup Y_2,..,Y_n|\)-ary predicate; both \( p_1^\star \) and \( p_2^\star \) are auxiliary predicates not introduced so far.

2. A TGD of the form \( \varphi(X) \rightarrow \exists Y p_1(X, Y_1),..,p_n(X, Y_n) \) is replaced by
   \[
   \varphi(X) \rightarrow \exists Y p^\star(X, Y)
   \]
   \[
   p^\star(X, Y) \rightarrow p_1(X, Y_1)
   \]
   \[
   \vdots
   \]
   \[
   p^\star(X, Y) \rightarrow p_n(X, Y_n),
   \]
   where \( p^\star \) is an \(|X \cup Y|\)-ary auxiliary predicate not introduced so far.

Now, from \( N_1(\sigma) \), we obtain \( N(\sigma) \) by applying the following: for each \( \sigma' \in N_1(\sigma) \), assuming that \( \text{frontier}(\sigma') = X \) and \( \{Y_1,..,Y_n\} \) are the existentially quantified variables of \( \sigma', \sigma' \) is replaced by

\[
\text{body}(\sigma') \rightarrow \exists Y_1 p_1^\sigma(X, Y_1)
\]
\[
p_1^\sigma(X, Y_1) \rightarrow \exists Y_2 p_2^\sigma(X, Y_1, Y_2)
\]
\[
p_2^\sigma(X, Y_1, Y_2) \rightarrow \exists Y_3 p_3^\sigma(X, Y_1, Y_2, Y_3)
\]
\[
\vdots
\]
\[
p_{n-1}^\sigma(X, Y_1,..,Y_{n-1}) \rightarrow \exists Y_n p_n^\sigma(X, Y_1,..,Y_n)
\]
\[
p_n^\sigma(X, Y_1,..,Y_n) \rightarrow \text{head}(\sigma),
\]
where, for each \( i \in [n] \), \( p_i^\sigma \) is an \((|X|+i)|\)-ary auxiliary predicate not introduced so far.

Finally, given a set \( \Sigma \) of DTGDS, \( N(\Sigma) \) is defined as \( \bigcup_{\sigma \in \Sigma} N(\sigma) \).

© YYYY ACM 0362-5915/YYYY/01-ARTA $15.00
DO 1: http://dx.doi.org/10.1145/0000000.0000000
ACM Transactions on Database Systems, Vol. V, No. N, Article A, Publication date: January YYYY.
B. PROOFS FROM SECTION 4

B.1. Proof of Theorem 4.1

We provide a polynomial-time reduction to the problem of deciding whether a GNFO sentence is unsatisfiable, which in turn is in 2ExpTime [Bárány et al. 2011]. Our reduction consists of two steps: (1) first, we reduce our problem to the problem of UCQ answering under frontier-guarded DTGDs; and (2) we show how the constructed set of frontier-guarded DTGDs can be equivalently rewritten as a GNFO sentence. Consider a database $D$, a set $\Sigma$ of weakly-frontier-guarded DTGDs, and a UCQ $Q$. Our reduction follows:

**Step 1.** The key idea is to exploit the fact that whenever the (disjunctive) chase applies a DTGD, by definition, in every model the non-affected variables (i.e., variables occurring at the non-affected positions) are mapped to constants of $C$. It is therefore possible to partially ground the DTGDs in such a way that the non-affected variables are replaced by all possible combinations of constants, yielding an equivalent set of frontier-guarded DTGDs. However, such a transformation would be exponential in the worst case. To overcome this difficulty we adapt a technique proposed in [Baget et al. 2011b] that allows us to simulate partial groundings with only a polynomial blow-up.

Roughly speaking, instead of explicitly replacing the non-affected variables by constants, we extend each predicate of the underlying schema in such a way that it contains all $m$ constants of the database in its first $m$ positions, and a grounding of its $n$ non-affected variables in the next $n$ positions. Using a set of $\text{sch}(|\Sigma|) \cdot n \cdot m$ inclusion dependencies, each propagating one of the constants to one of the non-affected variables, we can simulate the partial grounding as desired. Using this procedure we can convert a weakly-frontier-guarded set of DTGDs into a set of frontier-guarded DTGDs in polynomial time. Let us now formalize the above informal description.

Given a DTGD $\sigma \in \Sigma$, $\text{nav}(\sigma)$ denotes the set of variables occurring in at least one position of nonaffected($\Sigma$). Fix also a bijection $\#_\sigma : \text{nav}(\sigma) \to \{1, \ldots, |\text{nav}(\sigma)|\}$. Let $C_1, \ldots, C_m$ and $N_1, \ldots, N_n$ be variables of $V$ not occurring in $\Sigma$, where $m$ is the number of constants $c_1, \ldots, c_m$ appearing in $D$, and $n = \max_{\sigma \in \Sigma} \{|\text{nav}(\sigma)|\}$. Given a term $t$, let $\tau_\sigma(t) = N_{\#_\sigma(t)}$ if $t \in \text{nav}(\sigma)$; otherwise, let $\tau_\sigma(t) = t$. The mapping $\tau_\sigma$ is extended to atoms as follows:

$$\tau_\sigma(p(t_1, \ldots, t_k)) = p(C_1, \ldots, C_m, N_1, \ldots, N_n, \tau_\sigma(t_1), \ldots, \tau_\sigma(t_k)).$$

Notice that $\tau_\sigma$ can naturally be defined for sets of atoms. We now define the function $\tau$, mapping DTGDs of $\Sigma$ to frontier-guarded DTGDs, as follows:

$$\tau(\sigma) = \tau_\sigma(\text{body}(\sigma)) \rightarrow \tau_\sigma(\text{head}(\sigma)).$$

Intuitively, the variables $C_1, \ldots, C_m$ will permanently hold the constants of $\text{dom}(D)$, while the variables $N_1, \ldots, N_n$ will be assigned a combination of those constants. We define the set $\Sigma'_C$ of frontier-guarded DTGDs as the set $\{\tau(\sigma) \mid \sigma \in \Sigma\} \cup \Sigma_C$, where $\Sigma_C$ consists of the following IDs: $\bigcup_{p \in \text{sch}(\Sigma), i \in [m], j \in [n]} \{\sigma_{i,j}^p\}$, with

$$\sigma_{i,j}^p = p(C_1, \ldots, C_m, N_1, \ldots, N_n, X_1, \ldots, X_k) \rightarrow p(C_1, \ldots, C_m, N_j^\ast, \ldots, N_k^\ast, X_1, \ldots, X_k),$$

where $k = \text{arity}(p)$, $N_j^\ast = C_i$, and $N_k^\ast = N_i$ for each $k \neq j$. Roughly, $\Sigma_C$ is responsible for generating all the possible combinations of the constants occurring in $D$. Finally, let $D' = \{\tau'(\bar{a}) \mid \bar{a} \in D\}$ and $Q' = \{\tau'(\bar{a}) \mid \bar{a} \in Q\}$, where

$$\tau'(p(t_1, \ldots, t_k)) = p(c_1, \ldots, c_m, c_{m+1}, \ldots, c_n, t_1, \ldots, t_k).$$

Let us clarify that the selection of the constant $c_1$ at positions $m + 1, \ldots, m + n$ is an arbitrary one since, due to the IDs of $\Sigma_C$, all the combinations of constants of $\text{dom}(D)$...
will eventually appear at those positions during the construction of the chase. By construction, $D \cup \Sigma \models Q$ iff $D' \cup \Sigma' \models Q'$; this completes the first step of our reduction.

Step 2. Observe that a frontier-guarded DTGD $\sigma$ of the form $\forall X (\varphi(X) \rightarrow \exists Y \psi(X, Y))$ can be equivalently rewritten as the sentence $\Phi_\sigma = \neg(\exists X (\varphi(X) \land \neg\exists Y \psi(X, Y)))$, which falls in GNFO since all the free variables of $\exists Y \psi(X, Y)$, that is, the variables of $X$, appear in the frontier-guard of $\varphi(X)$. Moreover, given a CQ $q$, $\neg q$ trivially falls in GNFO since in $q$ there are no free variables. From the above discussion, we conclude that the sentence $\Psi_{D', \Sigma', Q'} = (D' \land \bigwedge_{\sigma \in \Sigma'} \Phi_\sigma \land \bigwedge_{q \in Q'} \neg q)$ falls in GNFO. The claim follows since $D' \cup \Sigma' \models Q'$ iff $\Psi_{D', \Sigma', Q'}$ is unsatisfiable.

B.2. Proof of Lemma 4.2
Consider a database $D$, a set $\Sigma$ of frontier-guarded DTGDs, and a UCQ $Q$. For each $\sigma \in \Sigma$ of the form $\forall X (\varphi(X) \rightarrow \exists Y \psi(X', Y))$, where $X' \subseteq X$, we define

$$\tau_1(\sigma) = \forall X (\varphi(X) \rightarrow p_\sigma(X')) \quad \text{and} \quad \tau_2(\sigma) = \forall X' (p_\sigma(X') \rightarrow \exists Y \psi(X', Y)),$$

where $p_\sigma$ is an auxiliary $|X'|$-ary predicate not occurring in $sch(\Sigma)$. Assuming that $g_\sigma(X')$ is the frontier-guard of $\sigma$, $\tau_1(\sigma)$ can be equivalently rewritten as follows; in the sequel, $p'_\sigma$ is an auxiliary $|X'|$-ary predicate:

$$\forall X (\varphi(X) \rightarrow p_\sigma(X')) \equiv \neg\exists X (\varphi(X) \land \neg p_\sigma(X'))
\equiv (\neg\exists X (\varphi(X) \land p_\sigma(X'))) \land (\forall X' (g_\sigma(X') \land \neg p_\sigma(X') \rightarrow p'_\sigma(X')))
\equiv (\neg\exists X (\varphi(X) \land p_\sigma(X'))) \land (\forall X' (g_\sigma(X') \rightarrow p_\sigma(X') \lor p'_\sigma(X'))).

\begin{flalign*}
\Phi_\sigma^1 & = (\bigvee_{\sigma \in \Sigma} \Phi_\sigma^1) \quad \text{and} \quad \Phi_\sigma^2 & = (\bigwedge_{\sigma \in \Sigma} (\tau_2(\sigma) \land \Phi_\sigma^2)).
\end{flalign*}

We define the formulas:

$$\Psi_\Sigma^1 = \neg \bigvee_{\sigma \in \Sigma} \Phi_\sigma^1 \quad \text{and} \quad \Psi_\Sigma^2 = \bigwedge_{\sigma \in \Sigma} (\tau_2(\sigma) \land \Phi_\sigma^2).$$

It is not difficult to see that

$$D \cup \Sigma \models Q \quad \text{iff} \quad (D \land \Psi_\Sigma^1 \land \Psi_\Sigma^2) \models Q \quad \text{iff} \quad (D \land \Psi_\Sigma^2) \models (Q \lor \neg \Psi_\Sigma^1).$$

$\Psi_\Sigma^2$ is a set of guarded DTGDs, and thus is equivalent to a GFO sentence $\Gamma_\Sigma^2$; hence, $(D \land \Gamma_\Sigma^2)$ is a GFO sentence. Since $(Q \lor \neg \Psi_\Sigma^1)$ is a UCQ, the claim follows.

B.3. Proof of Theorem 4.6
The proof is via a reduction from the non-acceptance problem of an alternating exponential space Turing machine $M$ on the empty input (as in the proof of the previous result). Instead of giving the complete reduction, we are going to explain how the reduction employed in the proof of Theorem 4.5 can be adapted; notice that the database $D$ remains the same.
An atom

We assume a fixed order on $-V$ after replacing each variable tuple of terms $t$.

Q

We assume, w.l.o.g., that the CQs occurring in $Q$ valid state. To this aim, we add another disjunct

It should not be forgotten that we need to guarantee that each state-chain encodes a

unary and binary predicates.

The UCQ $Q$. Recall that the query $Q$, employed in the proof of Theorem 4.5, makes use of the predicate $\text{level}_0$, in order to ask whether a node is the root of a configuration tree, and of the predicate $s$, where $s \in S$, in order to ask whether a cell is marked with the state $s$ of $M$. However, those predicates are not available anymore, and the query $Q$ is modified as follows:

— An atom $\text{level}_0(X)$, occurring in $Q$, is replaced by the formula
  
  \[ \text{ctnode}(X) \land (\text{conf}\exists(X) \lor \text{conf}\forall(X)), \]

  which asks whether $X$ is a node of a configuration tree, and at the same time represents a configuration.

— We assume a fixed order on $S$, and let $\#_s$ be the position of $s \in S$ in this order. The length of the state-chain which encodes $s$ is $\#_s$. We use $\text{state}^i(X,Y)$ as a shorthand for $\exists Z_1 \ldots \exists Z_{i-1} (\text{state}(X,Z_1) \land \ldots \land \text{state}(Z_{i-1},Y))$, and we define the subquery:

  \[ \text{state}[i](X) \equiv \exists Y (\text{state}^i(X,Y) \land \text{end}(Y)), \]

  which asks whether there exists a state-chain of length $i$. An atom $\text{state}(X)$, where $s \in S$, occurring in $Q$, is simply replaced by the atom $\text{state}[\#_s](X)$.

It should not be forgotten that we need to guarantee that each state-chain encodes a valid state. To this aim, we add another disjunct $Q_{\text{length}}$ in $Q$, which is defined as

\[ \bigvee_{x \in \{\exists, \forall\}} \exists X \exists Y \exists Z \left( \text{ctnode}(X) \land \text{conf}\exists(X) \land \text{child}\exists(X,Y) \land \text{state}^{|S|+1}(Y,Z) \right). \]

The definition of $Q$ is now complete. This completes our proof.

B.4. Proof of Lemma 4.7

We assume, w.l.o.g., that the CQs occurring in $Q$ do not have variables in common. In the rest of the proof, let $t$ and $f$ be constants of $C$ not occurring in $D$. Given a tuple of terms $t$, we denote by $t_1$ the tuple of constant obtained after “freezing” $t$, i.e., after replacing each variable $V$ in $t$ with a new constant $c_V \in C$ not occurring in $\text{dom}(D) \cup \{t, f\}$. We are now ready to give the formal reduction.

The Database $D'$. The new database $D'$ is defined as follows:

\[ \{ p'(t,t) \mid p(t) \in D \} \]
\[ \cup \{ \text{true}(t), \text{false}(f), \text{or}(t,t), \text{or}(t,f), \text{or}(f,t), \text{or}(f,f) \} \]
\[ \cup \{ p'(t_1, f) \mid q \in Q \text{ and } p(t) \in \text{body}(q) \}. \]
The Set $\Sigma'$. The new set $\Sigma'$ of DTGDs is obtained from $\Sigma$ by replacing each atom of the form $p(X)$ occurring in a DTGD with the atom $p'(X, T)$, where $T$ is a new variable not occurring in $\Sigma$. For example, the DTGD $p(X, Y), s(Y, Z) \rightarrow \exists Wp(X, W), r(W)$ will be replaced by the DTGD $p'(X, Y, T), s'(Y, Z, T) \rightarrow \exists Wp'(X, W, T), r'(W, T)$.

The CQ $q$. Assume that $Q = q_1 \lor \ldots \lor q_n$. For a CQ $q_i \in Q$, we define $q_i[X_i]$ as the CQ obtained from $q_i$ by replacing each atom of the form $p(t)$ with the atom $p'(t, X_i)$, where $X_i$ is a new variable not occurring in $Q$. The CQ $q$ is defined as the query:

$$
\exists X_1 \ldots \exists X_n \exists Y_1 \ldots \exists Y_{n+1} \left( false(Y_1) \land \bigwedge_{q_i \in Q} (q_i[X_i] \land or(Y_i, X_i, Y_{i+1})) \land true(Y_{n+1}) \right).
$$

By construction, for each $q_i[X_i]$, there exists a homomorphism that maps $body(q_i[X_i])$ to the database $D'$. However, this fact does not imply that the query $q$ is trivially entailed by $\text{chase}(D', \Sigma')$ since, in order to satisfy the atom $true(Y_{n+1})$, at least one $X_i$ must be mapped to the constant $t$. Thus, by construction, the only way to entail $q$ is to map at least one subquery $q_i[X_i]$ to $\text{chase}(D', \Sigma')$ via a homomorphism $h$, and also $h(X_i) = t$. Notice that the only atoms in $\text{chase}(D', \Sigma')$ containing $t$ at the last position are the ones obtained from the original copy of $D$. Thus, a subquery $q_i \in Q$ is entailed by an instance $I \in \text{chase}(D, \Sigma)$ iff the corresponding instance $I'$ of $\text{chase}(D', \Sigma')$ entails $q$. Since the above construction is feasible in polynomial time, the claim follows.

C. PROOFS FROM SECTION 6

C.1. Proof of Theorem 6.2

Consider a database $D$, a weakly-guarded set $\Sigma$ of DTGDs with predicates of bounded arity, and an AUCQ $Q$. The proof is by reduction to satisfiability of LGFO sentences. More precisely, we are going to construct, in polynomial time, a set $\Sigma'$ of DTGDs such that $\Psi_{D, \Sigma, Q} = (D \land \Sigma' \land \neg Q)$ falls in LGFO, the arity of the underlying schema is increased by two (and therefore is bounded), and $D \cup \Sigma \models Q$ iff $\Psi_{D, \Sigma, Q}$ is unsatisfiable. Since the problem of deciding whether an LGFO sentence is in $\text{EXP\!TIME}$ in case of predicates of bounded arity [Grädel 1999], the claim follows.

Assume that $\text{dom}(D) = \{c_1, \ldots, c_n\}$, and let $\text{wgp}(\Sigma) \subseteq \text{sch}(\Sigma)$ be the predicates that appear in at least one $\sigma \in \Sigma$ as the predicate of the weak-guard of $\sigma$, that is, the atom that guards the body-variables of $\sigma$ that appear at affected positions. The set $\Sigma'$ is defined as follows. Given a DTGD $\sigma \in \Sigma$, let $\{W_1, \ldots, W_m\}$ be the set of variables occurring in at least one non-affected position in $\Sigma$. If $m = 0$ (which means that $\sigma$ is guarded), then let $\hat{\sigma} = \sigma$; otherwise, assuming that $\sigma$ is of the form $\varphi(X) \rightarrow \exists Y \varphi(X, Y)$, $g(t) \in body(\sigma)$ is the weak-guard of $\sigma$, and $\varphi(X)^\sim = \varphi(X) \setminus \{g(t)\}$, we define $\sigma$ as

$$
\left( \bigwedge_{1 \leq i < j \leq m} \hat{g}(W_i, W_j, t) \land \varphi(X)^\sim \right) \rightarrow \exists Y \varphi(X, Y).
$$

Moreover, for each $k$-ary predicate $p \in \text{wgp}(\Sigma)$, we define the TGD $\sigma_p$ as follows:

$$
p(X_1, \ldots, X_k) \rightarrow \bigwedge_{1 \leq i, j \leq n} \hat{p}(c_i, c_j, X_1, \ldots, X_k).
$$

The desired set of DTGDs is defined as

$$
\Sigma' = \bigcup_{\sigma \in \Sigma} \{\hat{\sigma}\} \cup \bigcup_{p \in \text{wgp}(\Sigma)} \{\sigma_p\}.
$$
It is easy to verify that $D \cup \Sigma \models Q$ iff $D \cup \Sigma' \models Q$, which in turn implies that $D \cup \Sigma \models Q$ iff $\Psi_{D,\Sigma, Q}$ is unsatisfiable. Intuitively speaking, a clique (with self-loops) of all the constants of $\text{dom}(D)$ is attached to all weak-guards of $\Sigma$, and all the variables at non-affected positions must map to an element of such a clique. As these non-affected variables can, by definition, map only to constants of $\text{dom}(D)$, one can see the above construction as a simulation of a partial grounding of $\Sigma$.

It remains to show that $\Psi_{D,\Sigma, Q}$ falls in LGFO; in fact, we are going to show that $\Psi_{D,\Sigma, Q}$ is “almost” an LGFO sentence which, as we explain below, it suffices for our purposes. Since $D$ is a conjunction of ground atoms, it immediately falls in GFO, and thus in LGFO. The set $\bigcup_{p \in \text{wgp} (\Sigma)} \{ \sigma_p \}$ trivially forms a GFO (and thus an LGFO) sentence. A DTGD $\sigma$ as above can be rewritten as

$$\forall X \forall W_1 \ldots \forall W_m \left( \bigwedge_{1 \leq i < j \leq m} \hat{g}(W_i, W_j, t) \rightarrow (\varphi(X)^{−} \rightarrow \exists Y \psi(X, Y)) \right).$$

The above sentence does not fall in LGFO due to the fact that the existentially quantified variables of $Y$ are not necessarily loosely-guarded. However, the satisfiability algorithm proposed in [Grädel 1999] is able to treat sentences which are “almost” LGFO sentences as the one above, even if the existentially quantified variables are not loosely-guarded, without increasing the complexity — this is explicitly stated in a remark on page 8 of [Grädel 1999]. Finally, by hypothesis, the query $Q$ is acyclic, which implies that there exists a UCQ which is equivalent to $Q$ and falls in GFO, and hence in LGFO [Gottlob et al. 2003]. It is clear that we have polynomially reduced our problem into a problem which is in EXPTIME, and the claim follows.

C.2. Proof of Theorem 6.6

The proof is by a reduction from the non-acceptance of an alternating linear space Turing machine $M$ on input $I$. Let $M = (S, \Lambda, \delta, s_0)$, where $S = S_{\forall} \uplus S_{\exists} \uplus \{ s_a \} \uplus \{ s_r \}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, $\Lambda = \{ 0, 1, \uplus \}$ is the tape alphabet with $\uplus$ be the blank symbol, $\delta : S \times \Lambda \rightarrow (S \times \Lambda \times \{-1, 0, +1\})^2$ is the transition function, and $s_0 \in S$ is the initial state. We make the same assumptions for $M$ as in the proof of Theorem 6.4, with the difference that now $M$ uses exactly $n$ tape cells, where $n = |I|$. We are going to construct a database $D$, a set $\Sigma$ of DTGDs, and an AUCQ $Q$ such that $D \cup \Sigma \models Q$ iff $M$ rejects $I$, where $N(\Sigma)$ is a fixed set of DIDs, and $\text{sch}(N(\Sigma))$ consists of unary and binary predicates. By Lemma 4.7, the obtained lower bound holds even for ACQs and predicates of arity at most three. The idea of the proof is along the lines of the one of Theorem 6.4, i.e., to construct trees, which encode possible computation trees of $M$ on the input string $I$, by chasing $D$ with $\Sigma$, and then exploit $Q$ to check their consistency. On each configuration node $v$, which represents the configuration $C_v$ of $M$, we attach a chain of length $n$, which mimics the tape in $C_v$, and also a chain of length at most $|S|$, which encodes the state of $C_v$. The formal construction of $D, \Sigma$ and $Q$ follows.

The Database $D$. Let $D = \{ \text{conf}_3(c) \}$, where $c \in C$ is a special constant which represents the initial configuration (which is, by assumption, existential).

The Set $\Sigma$. The predicates that we are going to use are self-explanatory, and thus we proceed with the construction of $\Sigma$ without describing the predicates of $\text{sch}(\Sigma)$.

— Each configuration has two successor configurations, such that a universal configuration is followed by existential configurations, and vice-versa. Note that for existential configurations, we generate two models for the successors, whereas for universal
configurations, both are forced to appear in the same model:

\[ \text{conf}_\exists(X) \rightarrow \exists Y \ \text{succ}_1(X, Y), \ \text{conf}_\forall(Y) \lor \exists Y \ \text{succ}_2(X, Y), \ \text{conf}_\forall(Y), \]

\[ \text{conf}_\forall(X) \rightarrow \exists Y \exists Z \ \text{succ}_1(X, Y), \ \text{conf}_\exists(Y), \ \text{succ}_2(X, Z), \ \text{conf}_\exists(Z). \]

— Each configuration has a cell- and state-chain which simulates its tape and encodes its state, respectively:

\[ \text{conf}_\exists(X) \rightarrow \exists Y \ \text{cell}(X, Y), \ \text{for each } x \in \{\exists, \forall\}, \]

\[ \text{cell}(X, Y) \rightarrow \exists Z \ \text{cell}(Y, Z) \lor \text{end}(Y), \]

\[ \text{conf}_\forall(X) \rightarrow \exists Y \ \text{state}(X, Y), \ \text{for each } x \in \{\exists, \forall\}, \]

\[ \text{state}(X, Y) \rightarrow \exists Z \ \text{state}(Y, Z) \lor \text{end}(Y). \]

— Finally, we guess the content of each cell and the cursor position:

\[ \text{cell}(X, Y) \rightarrow \bigvee_{\lambda \in \Lambda} \lambda(Y), \]

\[ \text{cell}(X, Y) \rightarrow \text{cursor}(Y) \lor \text{notCursor}(Y). \]

The construction of \( \Sigma \) is now completed. It is easy to see that \( \Sigma \) does not depend on \( M \), and also that \( N(\Sigma) \) is a set of DIDs.

The AUCQ \( Q \). We now proceed with the construction of \( Q \), which ensures that each model of \( D \cup \Sigma \) encodes an invalid computation tree. Roughly, \( Q \) consists of the following disjuncts:

1. \( Q_{\text{initial}} \) for checking that in the initial configuration the tape does not contain the input string \( I = a_1 \ldots a_n \), the state is not \( s_0 \), and the cursor is not at the first cell;
2. \( Q_{\text{length}} \) for checking that cell-chains are of length other than \( n \), and state-chains are of length greater than \( |S| \);
3. \( Q_{\text{cursor}} \) for checking that more than one cells are pointed by the cursor;
4. \( Q_{\text{trans}} \) for checking that the transition function of \( M \) is violated; and
5. \( Q_{\text{inertia}} \) for checking that the tape cells not under the cursor do not keep their old value during a transition.

We assume a fixed order on \( S \). Given a state \( s \in S \), let \( \#_s \) be its position in this order. The length of the state-chain which encodes \( s \) is \( \#_s \). For a predicate \( p \in \{\text{cell}, \text{state}\}, \)

\[ p'(X, Y) \] is used as a shorthand for \( \exists Z_1 \ldots \exists Z_{i-1} (p(X, Z_1) \land \ldots \land p(Z_{i-1}, Y)) \). A useful subquery that we are going to use is

\[ \text{length}[p]_k(X) \equiv \exists Y \ (p^k(X, Y) \lor \text{end}(Y)) \]

stating that there exists a \( p \)-chain of length \( k \). Having all the necessary ingredients in place, we are now ready to formalize the components of \( Q \).

1. \( Q_{\text{initial}} \) is defined as (recall that \( c \in C \) represents the initial configuration)

\[ \bigvee_{i \in [n]} \bigvee_{a \in \Lambda \setminus \{a_s\}} \exists X \ (\text{conf}_\exists(c) \land \text{cell}^i(c, X) \land a(X)) \lor \]

\[ \bigvee_{s \in S \setminus \{s_0\}} (\text{conf}_\exists(c) \land \text{length}[	ext{state}\#_s](c)) \lor \]

\[ \bigvee_{1 < i < n} \exists X \ (\text{conf}_\exists(c) \land \text{cell}^i(c, X) \land \text{head}(X)). \]
(2) $Q_{\text{length}}$ is defined as
\[ \bigvee_{1 \leq i < n} \exists X \text{ length[}cell\text{]}_i(X) \lor \exists X \forall Y \text{ cell}^{i+1}(X, Y) \lor \exists Y \forall X \text{ state}^{i+1}(X, Y). \]

(3) $Q_{\text{cursor}}$ is defined as
\[ \bigvee_{1 \leq i < j \leq n} \exists X \exists Y \exists Z \big( \text{cell}^i(X, Y) \land \text{cell}^j(X, Z) \land \text{cursor}(Y) \land \text{cursor}(Z) \big). \]

(4) In the definition of $Q_{\text{trans}}$, we use the function $f : S \rightarrow \{ \exists, \forall \}$, where $f(s) = \exists$ if $s \in S_3$; otherwise, $f(s) = \forall$. Moreover, we use the binary operator $\otimes$, where $s \in S$, which is defined as $\vee$, if $s \in S_3$, and as $\land$, if $s \in S_\forall$. Recall that $\delta_\forall$ and $\delta_\exists$ denote the complement of $\delta$ relative to universal and existential, respectively; for details, see the proof of Theorem 4.5. $Q_{\text{trans}}$ is defined as
\[ \bigvee_{(s, a) \rightarrow (s_1, a_1, d_1), (s_2, a_2, d_2) \in \delta} \big( \text{trans}[\text{state}]_{\#_1}(X) \land \text{length[}state\text{]}_{\#_2}(X) \land \text{cell}^i(X, Y) \land \text{cursor}(Y) \land a(Y) \land \text{successor}(X, Z) \land \text{length[}state\text{]}_{\#_2}(Z) \land \text{cell}^j(Z, V) \land a_i(V) \land \text{cell}^{i+1}(Z, W) \land \text{cursor}(W) \big). \]

(5) Finally, $Q_{\text{inertia}}$ is defined as
\[ \bigvee_{x \in \{ \exists, \forall \}} \bigvee_{(a, a') \in \Lambda \times \Lambda, a \neq a' \in [n]} \left( \text{successor}(X, Z_1) \lor \text{successor}(X, Z_2) \land \text{cell}^i(X, Z_1) \land \text{cell}^j(Y, Z_2) \land \text{notCursor}(Z_1) \land a(Z_1) \land a'(Z_2) \right). \]

This completes the construction of $Q$. It is easy to verify that each disjunct of $Q$ is an acyclic query.

By construction, and the assumption that a rejecting configuration does not have a successor configuration, while an accepting configuration has only itself as a successor, it is not difficult to see that $D \cup \Sigma \models Q$ if $M$ rejects $I$ (see the argument given in the proof of Theorem 5.1 which shows that the employed construction is correct).

D. PROOFS FROM SECTION 7

D.1. Proof of Lemma 7.3

Let $D = \{ q_1, \ldots, q_n \}$, and $q = \exists X p(X)$. Before we proceed further, we need to establish the following auxiliary claim:

**Claim D.1.** It holds that,
\[ \text{mods}(D, \Sigma) = \left\{ \bigcup_{i \in [n]} M_i \mid (M_1, \ldots, M_n) \in \text{mods}(\{q_1\}, \Sigma) \times \cdots \times \text{mods}(\{q_n\}, \Sigma) \right\}. \]

**Proof.** (\(\subseteq\)) Fix $M \in \text{mods}(D, \Sigma)$. Clearly, for each $i \in [n]$, $M_i \in \text{mods}(\{q_i\}, \Sigma)$. Thus, $(M_1, \ldots, M_n) \in \text{mods}(\{q_1\}, \Sigma) \times \cdots \times \text{mods}(\{q_n\}, \Sigma)$ which implies that $M \in M$.

(\(\supseteq\)) Conversely, fix a tuple $(M_1, \ldots, M_n) \in \text{mods}(\{q_1\}, \Sigma) \times \cdots \times \text{mods}(\{q_n\}, \Sigma)$, and let $M = \bigcup_{i \in [n]} M_i$. To show that $M \in \text{mods}(D, \Sigma)$, it suffices to show that, whenever for a DTGD $\sigma \in \Sigma$ there exists a homomorphism $h$ such that $h(\text{body}(\sigma)) \subseteq M$, then
Let us now conclude the proof of our lemma.

(⇒) Assume that, for each $i \in [n]$, $\{a_i\} \cup \Sigma \not\models q$, and thus there exists $M_i \in \text{mods}(a_i, \Sigma)$ such that $M_i \not\models q$. Therefore, $(M_1 \cup \ldots \cup M_n) \not\models q$, and by the above claim we get that $D \cup \Sigma \not\models q$.

(⇐) By hypothesis, there exists $i \in [n]$ such that $\{a_i\} \cup \Sigma \models q$. Consequently, for each $M \in \text{mods}(\{a_i\}, \Sigma)$, $M \models q$. Hence, by the above claim, each instance of $\text{mods}(D, \Sigma)$ entails $q$, and the claim follows.

D.2. Proof of Lemma 7.6

Let $\Sigma_f = \{\rho_1, \ldots, \rho_n\}$. By definition of the tree of a model, $\text{trees}(a, \Sigma_f)$ is isomorphic to the set

$$T = \bigcup_{i \in [n]} \text{trees}(a, \Sigma_f) \times \ldots \times \text{trees}(a, \Sigma_f),$$

where $\psi_{i \in [n]} T_i$ is the rooted tree obtained from the disjoint union of $T_1, \ldots, T_n$ after merging the root nodes (see Figure 10). We are now ready to show our lemma.

(⇒) Assume that for each $\rho \in \Sigma_f$ there exists $T \in \text{trees}(a, \Sigma_f)$ such that $T \not\models q$. Therefore, there exists $T' \in T$ such that $T' \not\models q$. Clearly, there exists $T'' \in \text{trees}(a, \Sigma_f)$ isomorphic to $T'$. Assuming that $T'' = \text{tree}(M)$, for some $M \in \text{mods}(\{a\}, \Sigma_f)$, $M \not\models q$; thus, $\{a\} \cup \Sigma_f \not\models q$.

(⇐) This direction follows immediately.

D.3. Proof of Lemma 7.7

(⇒) The claim is shown by giving a proof-tree of some $a \in D$ and $\Sigma$ which is valid w.r.t. $q$; assume that $q = \exists X \rho(X)$. By Lemmas 7.3 and 7.6, there exist $a \in D$ and $\rho \in \Sigma_f$ such that in every $T \in \text{trees}(a, \Sigma_f)$ there exists a path $\pi_T$ from the root to a node $u$ which is labeled by $h(p(X))$, where $h$ is a homomorphism, and the first edge $e_T$ of $\pi_T$ is labeled by a rule $\rho$ of the form $b \rightarrow b_1 \lor b_2$ or $b \rightarrow b_1$. Let us now construct a rooted binary tree $T'' = (N, E, \lambda_1, \lambda_2)$; initially, $N = \{v\}$, $E = \emptyset$, $\lambda_1 = \{v \rightarrow a\}$, and $\lambda_2 = \emptyset$. Clearly, $\rho\text{-trees}(a, \Sigma_f)$ can be partitioned into $S_1$ and $S_2$ such that, for each $T \in S_1$, if $e_T = (u, w)$ and $h$ maps $b$ into $a$ (which is the label of $u$), then $w$ is labeled by $h(b)$.

For each $i \in \{1, 2\}$, assume that the second node of $\pi_T$, for each $T \in S_i$, is labeled by $\bar{g}_i$. Let $N = N \cup \{w_i, w_{i+1}\}$, $E = E \cup \{(v, w_i)\}_{i \in \{1, 2\}}$, $\lambda_1 = \lambda_1 \cup \{w_i \rightarrow \bar{g}_i\}_{i \in \{1, 2\}}$, and $\lambda_2 = \lambda_2 \cup \{(v, w_i) \rightarrow \rho\}_{i \in \{1, 2\}}$. Due to the fact that $\{\bar{g}_i\} \cup \Sigma_f \models q$, for each $i \in [2]$, Lemma 7.6 can be recursively applied finitely many times as described above, and eventually $T''$ is constructed. The desired proof-tree is obtained from $T''$ by replacing the skolem terms occurring in $T''$ with distinct nulls of $N$. 

ACM Transactions on Database Systems, Vol. V, No. N, Article A, Publication date: January YYYY.
(⇐) By hypothesis, there exists \( a \in D \) and a proof-tree \( T \) of \( a \) and \( \Sigma \) which is valid w.r.t. \( q \). It is possible to construct from \( T \) a rooted binary tree \( T' \) such that the nodes are labeled by atoms with skolem terms (instead of nulls), the edges are labeled by rules of \( \Sigma_f \), and the structural properties of \( T \) are preserved. Assume that the label of each outgoing edge of the root of \( T' \) is \( \rho \). By Lemmas 7.3 and 7.6, it suffices to show that each \( \rho \)-tree of \( a \) and \( \Sigma_f \) entails \( q \). This follows from the fact that, for each \( T'' \in \rho \)-trees(\( a, \Sigma_f \)), there is a path from the root to a leaf of \( T'' \) that can be mapped into \( T'' \).

D.4. Normalization Procedure

Consider a linear DTGD \( \sigma \). First, we construct the set \( N_1(\sigma) \) by exhaustively applying the following two replacement rules, starting from \( \sigma \), until each assertion is either a single-atom-head TGD, or a DTGD with a disjunction of two atoms in its head:

1. A linear DTGD \( p(X) \rightarrow \bigvee_{i=1}^n \exists Y_i (p_i(X,Y_i), \ldots, p_k(X,Y_k)) \) is replaced by
   \[
   p(X) \rightarrow p_1(X) \lor p_2(X) \\
   p_1(X) \rightarrow \exists Y_1 p_1(X,Y_1) \\
   \vdots \\
   p_n(X) \rightarrow \exists Y_n p_n(X,Y_n)
   \]

   where \( p_1 \) and \( p_2 \) are \( |X| \)-ary auxiliary predicates not introduced so far.

2. A linear TGD \( p(X) \rightarrow \exists Y p_1(X,Y_1), \ldots, p_n(X,Y_n) \) is replaced by
   \[
   p(X) \rightarrow \exists Y_1 p_1(X,Y_1) \\
   \vdots \\
   p(X) \rightarrow \exists Y_n p_n(X,Y_n).
   \]

Now, from \( N_1(\sigma) \), we obtain \( N(\sigma) \) by employing the construction presented in Section 2 for transforming a set of DTGDs into a set of DTGDs with only one occurrence of an existentially quantified variable. Finally, given a set \( \Sigma \) of DTGDs, \( N(\Sigma) \) is defined as \( \bigcup_{\sigma \in \Sigma} N(\sigma) \).

D.5. Proof of Theorem 7.15

Let \( D, \Sigma \) and \( Q \) be the database, the set of DIDs, and the UCQ employed in the proof of Theorem 6.4, where it is shown that ACQ answering under DID is 2ExpTime-hard. Our goal is to construct in polynomial time a set \( \Sigma' \) of multi-linear DTGDs, which depends on \( \Sigma \) and \( Q \), such that \( D \cup \Sigma \models Q \) iff \( D \cup \Sigma' \models q \), where \( q = p^* \) with \( p^* \) be an auxiliary 0-ary predicate not occurring in \( \text{sch}(\Sigma) \). In fact, our intention is to obtain \( \Sigma' \) by adding to \( \Sigma \) the TGD \( \varphi \rightarrow p^* \), for each disjunct \( \varphi \) of \( Q \). However, to ensure that the added TGDs are multi-linear, we first need to modify the query \( Q \).

Let us start by explaining why we cannot directly add the disjuncts of \( Q \) to \( \Sigma \). Consider, for example, the disjunct \( Q_{\text{inertia}} \)

\[
\bigvee_{(s,s') \in S \times S} \bigvee_{(a,a') \in A \times A, a \neq a'} \bigvee_{i \in \{1,2\}} \bigvee_{j \in \{0,1\}} \exists X^{n} \exists \exists \exists \exists E \exists P \exists N_1 \exists N_2 \exists N'_1 \exists N'_2 (\text{conf}[s](X^n, a, 0, T, P, N_1, N_2) \land \text{conf}[s'](X^n, a', j, N_i, T, N'_1, N'_2)),
\]

which checks that the tape cells not under the cursor keep their old values during a transition. To perform such a check we just need to have access to a certain configura-
tion and to one of its subsequent configurations. However, in an atom \( \text{conf}[s](\cdot) \) we store more information than necessary, namely the previous configuration, and both subsequent configurations. This is precisely the reason why the atoms of \( Q_{\text{inertia}} \) contain different variables, and its disjuncts will give rise to TGDs that are not multi-linear. Thus, we need to project out the useless variables via some (non-disjunctive) IDs, and rewrite the disjuncts of \( Q \) analogously. The projection IDs, which are part of \( \Sigma' \), are

\[
\text{conf}[s](X^n, C, H, T, P, N_1, N_2) \rightarrow \text{conf}_p[s](X^n, C, H, T, P)
\]

\[
\text{conf}[s](X^n, C, H, T, P, N_1, N_2) \rightarrow \text{conf}_l[s](X^n, C, H, T, N_1)
\]

\[
\text{conf}[s](X^n, C, H, T, P, N_1, N_2) \rightarrow \text{conf}_r[s](X^n, C, H, T, N_2).
\]

The subscripts \( P, L \) and \( R \) stand for previous, (next-)left and (next-)right, respectively. Now, the query \( Q_{\text{inertia}} \) can be rewritten as follows:

\[
\bigvee_{x \in \{L,R\}} \bigvee_{(s,s') \in S \times S} \bigvee_{(a,a') \in \Lambda \times \Lambda, a \neq a'} \bigvee_{j \in \{0,1\}} (\text{conf}_x[s](X^n, a, 0, T, N) \land \text{conf}_p[s'](X^n, a', 0, N, T)).
\]

Observe that, for every disjunct \( \varphi \) of the above query, and for every pair of atoms \( a, b \) of \( \varphi \), \( \text{var}(a) = \text{var}(b) \); thus, the TGD \( \varphi \rightarrow p^* \) is multi-linear. The same approach can be applied also for \( Q_{\text{trans}} \) and \( Q_{\text{inertia}} \), without violating the multi-linearity of the added TGDs. However, this is not true for \( Q_{\text{move}} \). After rewriting \( Q_{\text{move}} \) as above, we get

\[
\bigvee_{x \in \{L,R\}} \bigvee_{0 < i, j \leq n} \bigvee_{(s,s') \in S \times S} (c_1, c_2, c_3) \in \Lambda^3
\]

\[
\left( \text{conf}_x[s](X^{n-i}, 0, 1^{i-1}, c_1, 1, T, N) \land \text{conf}_p[s'](X^{n-i}, 0, 1^{j-1}, c_2, 0, N, T) \right) \land
\]

\[
\left( \text{conf}_x[s](Y^{n-j}, 1, 0^{j-1}, c_1, 1, T, N) \land \text{conf}_p[s'](Y^{n-j}, 1, 0^{i-1}, c_3, 0, N, T) \right),
\]

and the TGD \( \varphi_1, \varphi_2 \rightarrow p^* \) is not multi-linear due to the variables \( X^{n-i} \) and \( Y^{n-j} \). Nevertheless, since we just need to join the variables \( T \) and \( N \), it suffices to add to \( \Sigma \) the multi-linear TGDs \( \varphi_1 \rightarrow \text{aux}_1(T, N), \varphi_2 \rightarrow \text{aux}_2(T, N) \) and \( \text{aux}_1(T, N), \text{aux}_2(T, N) \rightarrow p^* \), where \( \text{aux}_1 \) and \( \text{aux}_2 \) are auxiliary predicates not occurring in \( \text{sch}(\Sigma) \).

6.6. Proof of Theorem 7.16

The proof is by reduction from the acceptance problem of an alternating logarithmic space Turing machine \( M \) on input \( I = a_1 \ldots a_{|I|} \). Recall that logarithmic space Turing machines are equipped with a read-only input tape and a read/write work tape, where on the input tape only the cells which hold the input can be visited, while on the work tape only logarithmically many cells can be used. Let \( M = (S, \Sigma, \delta, s_0) \), where \( S = S_\text{q} \cup S_\text{L} \cup \{s_a\} \cup \{s_\text{r}\} \) is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, \( \Sigma = \{0,1,\sqcup\} \) is the tape alphabet with \( \sqcup \) be the blank symbol, \( \delta : S \times \Sigma \times \Sigma \rightarrow (S \times \Lambda \times \{-1,+1\})^2 \times \Lambda \times \{-1,+1\} \) is the transition function, and \( s_0 \in S \) is the initial state. We assume that \( M \) is well-behaved and never tries to read beyond its tape boundaries, and uses exactly \( n = \log |I|^k \) cells of the work tape, where \( k > 0 \).

We proceed to construct a database \( D \), a set \( \Sigma \) of DIDs, where \( \text{sch}(\Sigma) \) consists of unary predicates, and a CQ1, \( q \) such that \( D \cup \Sigma \models q \) if and only if \( M \) accepts. Each configuration

ACM Transactions on Database Systems, Vol. V, No. N, Article A, Publication date: January YYYY.
of $M$ is represented using a unary predicate of the form
\[ \text{conf}[s\#a_1 \ldots a_{|I|} x_1 \ldots x_{|I|} \#c_1 \ldots c_n y_1 \ldots y_n], \]
where $s \in S$, $a_1 \ldots a_{|I|}$ is the input tape, $x_1 \ldots x_{|I|} \in \{0,1\}^{|I|}$ indicates the position of the cursor on the input tape (e.g., $x_1 \ldots x_{i-1} x_i 1 x_{i+1} \ldots x_{|I|} = 0^{|I|-1}$ and $x_i = 1$ implies that the cursor is at the $i$-th cell), $c_1 \ldots c_n \in \Lambda^n$ is the work tape, and $y_1 \ldots y_n \in \{0,1\}^n$ indicates the position of the cursor on the work tape. Since the number of configurations of $M$ on $I$ is polynomial, we only need polynomially many predicates. We proceed with the construction of $D$, $\Sigma$, and $q$.

**The Database $D$.** Let $D$ be the database consisting of the single atom
\[ \text{conf}[s_0\#a_1 \ldots a_{|I|} 1 0 \ldots 0 \#1 \ldots 1 \ldots 1 1](c), \]
where $c$ is an arbitrary constant of $C$.

**The Set $\Sigma$.** With $\Sigma$ we encode the transitions of $\delta$. Let $f : \{0,1\} \rightarrow 2^{|I|}$ be the function such that $f(x)$, where $x \in \{0,1\}$, is the set of cells of the input tape which contain the symbol $x$. For each transition $(s,a,b) \rightarrow (s_1,a,b_1,-1)$, $(s_2,a,-b_2,0)$ of $\delta$, we add in $\Sigma$ the following DID's: for each $i \in f(a)$, $j \in [n-1]$ and $x_i \ldots x_{j-1}, x_{j+1} \ldots x_n \in \Lambda^{n-1}$, $p_1(X) \rightarrow p_2(X) \vee p_3(X)$, if $s \in S$, else, $p_1(X) \rightarrow p_2(X)$ and $p_1(X) \rightarrow p_3(X)$, where
\[
\begin{align*}
p_1 &= \text{conf}[s\#a_1 \ldots a_{i-1} a_{i+1} \ldots a_{|I|} 0 \ldots 0 \#1 \ldots 0 \# x_1 \ldots x_{j-1} x_{j+1} \ldots x_n 0 \ldots 0 1 0 \ldots 0] \\
p_2 &= \text{conf}[s_1\#a_1 \ldots a_{i-1} a_{i+1} \ldots a_{|I|} 0 \ldots 0 1 \# x_1 \ldots x_{j-1} b_1 x_{j+1} \ldots x_n 0 \ldots 0 1 0 \ldots 0] \\
p_3 &= \text{conf}[s_2\#a_1 \ldots a_{i-1} a_{i+1} \ldots a_{|I|} 0 \ldots 0 1 \# x_1 \ldots x_{j-1} b_2 x_{j+1} \ldots x_n 0 \ldots 0 1 0 \ldots 0].
\end{align*}
\]
Similar DID's are added to $\Sigma$ for all the other forms of transitions occurring in $\delta$. Finally, for each predicate of the form $\text{conf}[s \ldots ]$ introduced above, which represents an accepting configuration, we add the (non-disjunctive) ID
\[ \text{conf}[s \ldots ](X) \rightarrow \text{accept}(X). \]
This completes the construction of $\Sigma$.

**The CQ.** With $q$ we just need to ask whether an accepting configuration has been reached, i.e., $\exists X \text{accept}(X)$, where $X \in V$.

By construction, $D \cup \Sigma \models q$ iff $M$ accepts. It remains to show that the total number of DID's that we need to construct is polynomial w.r.t. $|I|$. Recall that $n = \log |I|^k$, where $k > 0$. Clearly,
\[ |\Sigma| \leq |I| \cdot (n - 1) \cdot 3^n \cdot 2 = |I| \cdot (\log |I|^k - 1) \cdot 3^\log |I|^k \cdot 2 \leq |I| \cdot (\log |I|^k - 1) \cdot |I|^{2k} \cdot 2, \]
and the claim follows.

**D.7. Proof of Theorem 7.17**
This result can be obtained by adapting the proof of Theorem 7.15. Since we have to simulate an alternating polynomial space (and not exponential space) Turing machine,
the polynomially many cells (assume that the machine uses \( n \) cells) can be encoded in the predicates, and thus the arity becomes fixed. More precisely, we use the predicates
\[
conf[x\#0\ldots01\ldots0]_{i-1} \quad \text{and} \quad conf[x\#0\ldots01\ldots0],
\]
where \( x \in \{P,L,R\} \), which correspond to the \( i \)-the cell of the encoded configuration. Now, it is straightforward to see how the set of multi-linear DTGDs given in the proof of Theorem 7.15 can be adapted, and the claim follows.

**E. PROOFS FROM SECTION 8**

Before giving the proofs that are missing from Section 8, let us first recall the semantics of \( ALCIT \). An \( ALCIT \)-interpretation \( \mathcal{I} \) is a pair \((\Delta_{x}, \mathcal{I})\), where \( \Delta_{x} \) is a non-empty set called domain, and \( \mathcal{I} \) is an interpretation function that maps every concept name \( A \) to a subset \( A^{\mathcal{I}} \) of \( \Delta_{x} \), and every role name \( R \) to a binary relation \( R^{\mathcal{I}} \) of \( \Delta_{x} \). \( \mathcal{I} \) is extended to complex concepts as follows:

\[
\neg \mathcal{I} := \Delta_{x} \setminus C^{\mathcal{I}}
\]
\[
(B \cap C)^{\mathcal{I}} := B^{\mathcal{I}} \cap C^{\mathcal{I}}
\]
\[
(B \cup C)^{\mathcal{I}} := B^{\mathcal{I}} \cup C^{\mathcal{I}}
\]
\[
(\exists R.C)^{\mathcal{I}} := \{x \mid \text{there exists } y \in \Delta_{x} \text{ such that } (x, y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}
\]
\[
(\forall R.C)^{\mathcal{I}} := \{x \mid \text{for all } y \in \Delta_{x}, (x, y) \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}
\]
\[
(\forall R^{-}.C)^{\mathcal{I}} := \{x \mid \text{for all } y \in \Delta_{x}, (y, x) \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}.
\]

An interpretation \( \mathcal{I} \) is a model of a TBox \( \mathcal{T} \) if \( B^{\mathcal{I}} \subseteq C^{\mathcal{I}} \) for all \( B \subseteq C \in \mathcal{T} \). An interpretation \( \mathcal{I} \) can be extended to ABoxes by demanding that \( \mathcal{I} \) maps every individual \( a \) to an element \( a^{\mathcal{I}} \in \Delta_{x} \). \( \mathcal{I} \) satisfies an assertion \( C(a) \) if \( a^{\mathcal{I}} \in C^{\mathcal{I}} \) and an assertion \( R(a,b) \) if \( (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} \). We say that \( \mathcal{I} \) is a model of an ABox \( \mathcal{A} \) if it satisfies all the assertions of \( \mathcal{A} \). Given an \( ALCIT \) KB \((\mathcal{T}, \mathcal{A})\), \( \mathcal{I} \) is a model of \( \mathcal{K} \), written \( \mathcal{I} \models \mathcal{K} \), if \( \mathcal{I} \) is a model of \( \mathcal{T} \) and \( \mathcal{A} \). It is now straightforward to define the problem of query answering under \( ALCIT \) KBs. A CQ \( q \) over an \( ALCIT \) KB \( \mathcal{K} \) is a conjunction of atoms of the form \( A(X) \) or \( R(X,Y) \), where \( A \in \text{N}_{C} \) and \( R \in \text{N}_{R} \). The answer to \( q \) is positive, written \( \mathcal{K} \models q \), if \( \mathcal{I} \models q \), for every model \( \mathcal{I} \) of \( \mathcal{K} \).

**E.1. Proof of Theorems 8.1 and 8.2**

**Upper Bounds.** The upper bounds for both theorems are obtained by reducing query answering under \( ALCIT \) to query answering under guarded DTGDs. First, we discuss how complex concepts can be eliminated from the given ABox by pushing them in the TBox. Such a reformulation of the given KB will allow us to consider the ABox as a relational database. Consider an \( ALCIT \) KB \( \mathcal{K} = (\mathcal{T}, \mathcal{A}) \). We construct an ABox \( \mathcal{A}' \) from \( \mathcal{A} \) by replacing each assertion \( C(a) \), where \( C \) is a complex concept, by \( C^+(a) \), where \( C^+ \) is an auxiliary concept name not occurring in \( \mathcal{A} \). Then, the TBox \( \mathcal{T}' \) is constructed by adding to \( \mathcal{T} \) an assertion of the form \( C^+ \subseteq C \), for each complex concept \( C \) occurring in \( \mathcal{A} \). It is easy to see that \((\mathcal{T}, \mathcal{A})\) and \((\mathcal{T}', \mathcal{A}')\) are equivalent w.r.t. UCQ answering. Let us now explain how an \( ALCIT \) TBox can be normalized in such a way that can be seen as a set of guarded DTGDs. In fact, we exploit the normalization procedure proposed in [Simancik et al. 2011] for \( ALCH \), i.e., \( ALCIT \) without inverse roles. An \( ALCIT \) TBox is in normal form if it only contains axioms of the form given on the left column of Table IV. It is implicit in [Simancik et al. 2011] that every \( ALCH \) TBox can be transformed in polynomial time into an \( ALCH \) TBox in normal form, which is
equivalent to $\mathcal{T}$ w.r.t. UCQ answering. This result can be straightforwardly extended to $\mathcal{ALCITH}$. From the above discussion, we immediately get the following auxiliary result:

**Lemma E.1.** Consider an $\mathcal{ALCITH}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. We can construct in polynomial time an $\mathcal{ALCITH}$ KB $\mathcal{K}' = (\mathcal{T}', \mathcal{A}')$ such that $\mathcal{A}'$ contains only concept and role names, and $\mathcal{K} \models Q$ iff $\mathcal{K}' \models Q$, for every UCQ $Q$.

As shown in Table IV, given an $\mathcal{ALCITH}$ TBox $\mathcal{T}$ in normal form, every axiom of $\mathcal{T}$ can be equivalently rewritten as a first-order sentence, which is either a guarded DTGD or a constraint of the form $\forall X (A_1(X) \land \ldots \land A_n(X) \rightarrow \bot)$, where $\bot$ denotes the truth constant $false$, which may lead to an inconsistency. Such inconsistencies can be encoded in the query by considering the left-hand side of the constraints as disjuncts of the given UCQ. Moreover, observe that these disjuncts will be BTWCQs and ACQs, and thus, if the given UCQ is of bounded treewidth (resp., acyclic), then the obtained UCQ remains of bounded treewidth (resp., acyclic). Now, an ABox that contains only concept and role names (and not complex concepts) can be naturally seen as a relational database. Therefore, from Lemma E.1 and the above discussion, we conclude the following: given an $\mathcal{ALCITH}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and a UCQ (resp., BTWUCQ, AUCQ) $Q$, we can construct in polynomial time a database $D_\mathcal{A}$, a set $\Sigma_\mathcal{T}$ of guarded DTGDs, and a UCQ (resp., BTWUCQ, AUCQ) $Q_\mathcal{T}$ such that $\mathcal{K} \models Q$ iff $D_\mathcal{A} \cup \Sigma_\mathcal{T} \models Q_\mathcal{T}$, and the next result follows:

**Proposition E.2.** UCQ (resp., BTWUCQ, AUCQ) answering under $\mathcal{ALCITH}$ KBs can be reduced to UCQ (resp., BTWUCQ, AUCQ) answering under guarded DTGDs in polynomial time.

Clearly, the above proposition, combined with our results on query answering under guarded DTGDs, implies the upper bounds stated in Theorems 8.1 and 8.2.

**Lower Bounds.** The desired lower bounds are inherited from Theorems 4.6, 5.1 and 6.6. In fact, the sets of DIDs employed in the proofs of the above results can be equivalently rewritten as DL axioms.

We first focus on the proof of Theorems 4.6 and 5.1; recall that the above theorems have exactly the same proof. The DIDs employed in this proof have one of the following

<table>
<thead>
<tr>
<th>$\mathcal{ALCITH}$ Axiom</th>
<th>First-Order Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 \land \ldots \land A_n \subseteq \bot$</td>
<td>$\forall X (A_1(X) \land \ldots \land A_n(X) \rightarrow \bot)$</td>
</tr>
<tr>
<td>$A_1 \land \ldots \land A_n \subseteq B_1 \lor \ldots \lor B_m$</td>
<td>$\forall X (A_1(X) \land \ldots \land A_n(X) \rightarrow B_1(X) \lor \ldots \lor B_m(X))$</td>
</tr>
<tr>
<td>$A \subseteq \exists R.B$</td>
<td>$\forall X (A(X) \rightarrow \exists Y R(X, Y) \land B(Y))$</td>
</tr>
<tr>
<td>$\exists R.A \subseteq B$</td>
<td>$\forall X (R(X, Y) \land A(Y) \rightarrow B(X))$</td>
</tr>
<tr>
<td>$\exists R.X \subseteq B$</td>
<td>$\forall X (A(X) \land X \subseteq B(X))$</td>
</tr>
<tr>
<td>$A \subseteq \forall R.X$</td>
<td>$\forall X (A(X) \land R(X, Y) \rightarrow B(Y))$</td>
</tr>
<tr>
<td>$R \subseteq S$</td>
<td>$\forall X \forall Y (R(X, Y) \rightarrow S(X,Y))$</td>
</tr>
<tr>
<td>$R^- \subseteq S$</td>
<td>$\forall X \forall Y (R(X, Y) \rightarrow S(X,Y))$</td>
</tr>
<tr>
<td>$R^- \subseteq S^-$</td>
<td>$\forall X \forall Y (R(X, Y) \rightarrow S(X,Y))$</td>
</tr>
</tbody>
</table>

Table IV: Normal form and first-order representation.
forms that correspond to a DL axiom:

\[ A(X) \rightarrow B_1(X) \lor \ldots \lor B_n(X) \equiv A \sqsubseteq B_1 \sqcup \ldots \sqcup B_n \]
\[ A(X) \rightarrow \exists Y R(X, Y) \equiv A \sqsubseteq \exists R.\top \]
\[ R(X, Y) \rightarrow A(Y) \equiv \exists R^-.\top \sqsubseteq A \]
\[ R(X, Y) \rightarrow S(X, Y) \equiv R \sqsubseteq S \]
\[ R(X, Y) \rightarrow S(Y, X) \equiv R^- \sqsubseteq S. \]

Clearly, an axiom of the form \( A \sqsubseteq B_1 \sqcup \ldots \sqcup B_n \) can be rewritten as

\[ A \sqsubseteq B_1 \sqcup B_{2..n} \]
\[ B_{2..n} \sqsubseteq B_2 \sqcup B_{3..n} \]
\[ \vdots \]
\[ B_{(n-2)..n} \sqsubseteq B_{n-2} \sqcup B_{(n-1)n} \]
\[ B_{(n-1)n} \sqsubseteq B_{n-1} \sqcup B_n, \]

and the next result follows:

**Proposition E.3.** Consider a DL \( \mathcal{L} \) that is powerful enough for expressing the following inclusion assertions:

\[ A_1 \sqsubseteq A_2 \sqcup A_3 \quad A \sqsubseteq \exists R.\top \quad \exists R^-\top \sqsubseteq A \quad R \sqsubseteq S \quad R^- \sqsubseteq S, \]

where \( A, A_1, A_2, A_3 \) are concept names and \( R,S \) are role names. Then, \( \text{(BTW)} \text{UCQ answering under } \mathcal{L} \text{ is } 2\text{EXPTIME-hard in combined complexity.} \)

Let us now focus on the proof of Theorem 6.6. The DIDs employed in this proof have one of the following forms that correspond to DL axioms (possibly more than one):

\[ A(X) \rightarrow \exists Y R(X, Y) \equiv A \sqsubseteq \exists R.\top \]
\[ R(X, Y) \rightarrow A(Y) \equiv \exists R^-\top \sqsubseteq A \]
\[ R(X, Y) \rightarrow \exists Z S(Y, Z) \lor A(Y) \equiv \exists R^-\top \sqsubseteq B_1 \quad B_1 \sqsubseteq B_2 \sqcup A \quad B_2 \sqsubseteq \exists S.\top \]
\[ R(X, Y) \rightarrow A_1(Y) \lor A_2(Y) \lor A_3(Y) \equiv \exists R^-\top \sqsubseteq B \quad B \sqsubseteq A_1 \sqcup A_{23} \quad A_{23} \sqsubseteq A_2 \sqcup A_3, \]

and the next result follows:

**Proposition E.4.** Consider a DL \( \mathcal{L} \) that is powerful enough for expressing the following inclusion assertions:

\[ A_1 \sqsubseteq A_2 \sqcup A_3 \quad A \sqsubseteq \exists R.\top \quad \exists R^-\top \sqsubseteq A, \]

where \( A, A_1, A_2, A_3 \) are concept names. Then, \( \text{AUCQ answering under } \mathcal{L} \text{ is } \text{EXPTIME-hard in combined complexity, even when the TBox is fixed.} \)