Ontology-Mediated Queries Distributing over Components

Citation for published version:

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Ontology-Mediated Queries Distributing Over Components

Gerald Berger Andreas Pieris
Institute of Information Systems, TU Wien, Austria
{berger, pieris}@dbai.tuwien.ac.at

Abstract

Ontology-based data access is concerned with the problem of querying incomplete data sources in the presence of an ontology. A key notion in this setting is that of ontology-mediated query, which is a database query coupled with an ontology. An interesting issue is whether the answer to an ontology-mediated query can be computed by parallelizing it over the connected components of the database, i.e., whether the query distributes over components. This allows us to evaluate the query in a distributed and coordination-free manner. We investigate distribution over components for classes of ontology-mediated queries where the database query is a conjunctive query and the ontology is formulated using existential rules. For each such class, we syntactically characterize its fragment that distributes over components, and we study the problem of deciding whether a query distributes over components.

1 Introduction

Ontology-based data access (OBDA) has recently emerged as a promising application of knowledge representation and reasoning technologies in information management systems. The aim of OBDA is to facilitate access to data that is significantly heterogeneous and incomplete. This is achieved via an ontology that provides a unified conceptual view of the data, and makes it accessible via database queries formulated solely in the vocabulary of the ontology. The actual database query and the ontology can be seen as two components of one composite query, called ontology-mediated query [Bienvenu et al., 2014]. OBDA can then be realized as the problem of answering ontology-mediated queries.

A challenging and important topic for declarative database query languages, and in particular (extensions of) Datalog, is coordination-free evaluation. This has its roots in declarative networking [Loo et al., 2009], an approach where distributed computations are programmed using Datalog-based formalisms. In this setting, programs (or queries) are specified over a global schema, and are executed by multiple computing nodes over which the database is distributed. These nodes can perform local computations, and also communicate asynchronously with each other via messages. The model assumes that messages can never be lost but can be arbitrarily delayed. An intrinsic source of inefficiency in such systems are the global barriers raised by the need for synchronization in computing the result of queries. This has motivated a fruitful line of research for isolating classes of queries that can be evaluated in a coordination-free manner [Zinn et al., 2012; Ameloot et al., 2013; 2014; 2015; Alvaro et al., 2014].

It is quite natural to ask whether the results on coordination-free evaluation for declarative database query languages can be transferred to ontology-mediated queries. Undoubtedly, a positive answer to this question, apart from possible applications to declarative networking, will significantly contribute towards more efficient procedures for OBDA, since it will enable distributed ontology-mediated query evaluation in a coordination-free way. The goal of the present work is to initiate the study of coordination-free evaluation for ontology-mediated queries, and give strong indications that the answer to the above challenging question is affirmative.

Distribution Over Components. More concretely, we focus on the question whether the answer to an ontology-mediated query can be computed by parallelizing it over the (maximally connected) components of the database, i.e., whether the query distributes over components. In other words, given an ontology-mediated query $Q$, the question is whether for every database $D$ the answer to $Q$ over $D$, denoted $Q(D)$, coincides with $\bigcup_{1 \leq i \leq n} Q(D_i)$, where $D_1, \ldots, D_n$ are the components of $D$. In this case, $Q(D)$ can be computed without any communication over a network using a distribution where every computing node is assigned some of the connected components of the database, and every component is assigned to at least one computing node.

The notion of distribution over components has been introduced in [Ameloot et al., 2014], and explicitly considered for Datalog queries in [Ameloot et al., 2015]. It has been shown that connected Datalog, that is, the fragment of Datalog where all rule-bodies are connected, provides an effective syntax for Datalog queries that distribute over components, while the problem of deciding whether a Datalog query distributes over components is undecidable.

Aims and Objectives. As said, this work is concerned about ontology-mediated queries and their distribution over components. We focus on ontology-mediated queries where the database query is a conjunctive query and the ontology is for-
mulated using existential rules (a.k.a. tuple-generating dependencies or Datalog\(^\pm\) rules). It is widely accepted that existential rules form a natural and convenient way for modeling ontologies. This has led to a flurry of activity for designing ontology languages based on existential rules; see e.g., [Baget et al., 2011; Cali et al., 2012a; 2012b; Thomazo et al., 2012; Leone et al., 2012]. We consider several ontology-mediated query languages, where the ontology is modeled using either arbitrary existential rules (without syntactic restrictions), or one of the basic decidable classes of existential rules, namely guarded, linear, or sticky (more details are given in Section 2), and we perform similar analysis as the one in [Ameloot et al., 2015] for Datalog queries. In particular, we would like to understand whether connectedness is the right notion for characterizing the fragment of the query languages in question that distributes over components, and understand the complexity of checking distribution over components.

**Our Results.** Our results can be summarized as follows:

- We show that for all the ontology-mediated query languages \(Q\) in question, the fragment of \(Q\) that distributes over components has the same expressive power as the fragment of \(Q\) where both the existential rules and the conjunctive query are connected (Theorem 2). For the language where the ontology is expressed using arbitrary existential rules, similar ideas as the ones in [Ameloot et al., 2015] for Datalog queries can be applied. However, the situation changes once we focus on the languages based on guarded, linear, or sticky existential rules, since these languages are not powerful enough to express transitivity axioms, which can be used to compute the connected components of the input database.

- The problem of deciding distribution over components is undecidable for all the ontology-mediated query languages that capture Datalog; implicit in [Ameloot et al., 2015]. Interestingly, we show that for the languages based on guarded, linear, or sticky existential rules, the problem is decidable (Theorems 15, 16 and 18).\footnote{For guarded-based queries, this holds under the assumption that the conjunctive query is answer-guarded, i.e., it has an atom that contains all the answer variables.} Furthermore, for the languages based on linear and sticky rules, we obtain precise complexity results that range from \(\Pi_2^P\) up to coNEXPTIME.

- It turned out that our techniques and results have interesting consequences to ontology-mediated query languages based on Description Logics (Corollary 8, and Theorem 19), and to central database query languages, i.e., (unions of) conjunctive queries, and non-recursive Datalog queries (Corollary 11, and Theorems 20 and 21).

### 2 Preliminaries

#### Instances and Queries.
Let \(C\), \(N\), and \(V\) be pairwise disjoint countably infinite sets of constants, (labeled) nulls and (regular) variables (used in queries and dependencies), respectively. A schema \(S\) is a finite set of relation symbols (or predicates) with associated arity. We write \(R/n\) to denote that \(R\) has arity \(n\). A term is either a constant, null or variable. An atom over \(S\) is an expression \(R(t)\), where \(R\) is a relation symbol in \(S\) of arity \(n > 0\) and \(t\) is an \(n\)-tuple of terms. A fact is an atom whose arguments consist only of constants. An instance over \(S\) is a (possibly infinite) set of atoms over \(S\) that contain constants and nulls, while a database over \(S\) is a finite set of facts over \(S\). The active domain of an instance \(I\), denoted \(\text{dom}(I)\), is the set of all terms occurring in \(I\).

A query over \(S\) is a mapping \(q\) that maps every database \(D\) over \(S\) to a set of answers \(q(D) \subseteq \text{dom}(D)^n\), where \(n \geq 0\) is the arity of \(q\). If \(n = 0\), then \(q\) is a Boolean query, and we write \(q(D) = 1\) if \((\cdot) \in q(D)\), and \(q(D) = 0\) otherwise.

The usual way of specifying queries is by means of (fragments of) first-order logic. A conjunctive query (CQ) \(q\) over \(S\) is a conjunction of atoms of the form \(\exists \vec{y} \phi(\vec{x}, \vec{y})\), where \(\vec{x} \cup \vec{y}\) are variables of \(V\), that uses only predicates from \(S\). The free variables of a CQ are called answer variables. The evaluation of CQs over instances is defined in terms of homomorphisms; we assume the reader is familiar with the standard notion of homomorphism. Let \(I\) be an instance over \(S\). The evaluation of \(q\) over \(I\), denoted \(q(I)\), is the set of all tuples \(h(\vec{x})\) of constants such that \(h\) is a homomorphism from \(q\) to \(I\). Each schema \(S\) and CQ \(q = \exists \vec{y} \phi(\vec{x}, \vec{y})\) give rise to the n-ary query \(q_\vec{y}.S\) defined by setting, for every database \(D\) over \(S\), \(q_\vec{y}.S(D) = \{ c \in \text{dom}(D)^n \mid c \in q(D) \}\). Let CQ be the class of all queries definable by some CQ. Analogously, we define UCQ as the class of all queries definable by some union of conjunctive queries (UCQ), that is, a disjunction of CQs with the same answer variables.

Two queries \(q_1\) and \(q_2\) over \(S\) are equivalent, written \(q_1 \equiv q_2\), if, for every database \(D\) over \(S\), \(q_1(D) = q_2(D)\). A query language \(Q_2\) is at least as expressive as query language \(Q_1\), written \(Q_1 \preceq Q_2\), if, for every \(q_1 \in Q_1\) over schema \(S\), there is \(q_2 \in Q_2\) over \(S\) such that \(q_1 \equiv q_2\). \(Q_1\) and \(Q_2\) have the same expressive power, written \(Q_1 = Q_2\), if \(Q_1 \preceq Q_2\) and \(Q_2 \preceq Q_1\).

#### Tgds for Specifying Ontologies.
An ontology language is a fragment of first-order logic. Here, we focus on languages that are based on tuple-generating dependencies. A tuple-generating dependency (tgd) is a first-order sentence of the form \(\forall \vec{x} \forall \vec{y} (\phi(\vec{x}, \vec{y}) \rightarrow \exists \vec{z} \psi(\vec{x}, \vec{z}))\), where both \(\phi\) and \(\psi\) are conjunctions of atoms without nulls and constants. For simplicity, we write this tgd as \(\phi(\vec{x}, \vec{y}) \rightarrow \exists \vec{z} \psi(\vec{x}, \vec{z})\), and use comma instead of \(\land\) for conjoining atoms. We call \(\phi\) and \(\psi\) the body and head of the tgd, respectively, and write \(\text{sch}(\Sigma)\) for the set of predicates occurring in \(\Sigma\). An instance \(I\) satisfies the above tgd if the following holds: For every homomorphism \(h\) such that \(h(\phi(\vec{x}, \vec{y})) \subseteq I\), there is a homomorphism \(h'\) that extends \(h\) such that \(h'(\psi(\vec{x}, \vec{z})) \subseteq I\). I satisfies a set \(\Sigma\) of tgds, denoted \(I \models \Sigma\), if \(I\) satisfies every tgd in \(\Sigma\). Let TGD be the class of all (finite) sets of tgds.

#### Ontology-Mediated Queries.
An ontology-mediated query over \(S\) is a triple \((\Sigma, \Sigma, q)\), where \(\Sigma \in \text{TGD}\), \(q \in \text{CQ}\), and \(q\) is over \(\Sigma \cup \text{sch}(\Sigma)\). We call \(S\) the data schema. In general, ontology-mediated queries are defined for arbitrary ontology languages (not only TGD), and arbitrary query languages (not only CQ). In this work, we focus on tgd-based ontology languages and queries definable via CQs. \(S\) is explicitly mentioned in the specification of the ontology-mediated query to highlight that it is interpreted as a query over \(S\). The semantics of an ontology-mediated query is given in terms of certain answers. Let \((\Sigma, \Sigma, q)\) be an ontology-mediated query, where \(n\) is the arity of \(q\). The certain answers to \(q\) w.r.t. a
database \( D \) over \( S \) and \( \Sigma \) is the set of tuples \( \text{cert}_{q,\Sigma}(D) = \{ \bar{c} \in \text{dom}(D)^n \mid \bar{c} \in q(I), \forall I \text{ s.t. } I \supseteq D \text{ and } I \models \Sigma \} \). Recall that \( \text{cert}_{q,\Sigma}(D) \) coincides with the evaluation of \( q \) over the canonical instance of \( D \) and \( \Sigma \) that can be constructed by the chase; see e.g., [Calì et al., 2012b]. The chase adds new atoms to \( D \) as dictated by \( \Sigma \) until the final result, written \( \text{chase}(D, \Sigma) \), satisfies \( \Sigma \). When an existentially quantified variable must be satisfied the chase invents a new null.

**Ontology-Mediated Query Languages.** Every query \( Q = (S, \Sigma, q) \) can be semantically interpreted as a query \( q_Q \) over \( S \) by setting \( q_Q(D) = \text{cert}_{q,\Sigma}(D) \), for every \( S \)-database \( D \).\(^2\) Thus, we obtain a new query language, denoted \((\text{TGD}, \text{CQ})\), defined as the class of queries \( q_Q \), where \( Q \) is an ontology-mediated query. However, \((\text{TGD}, \text{CQ})\) is undecidable since, given a database \( D \) over \( S, \Sigma \in \text{TGD} \), an \( n \)-ary query \( q \in \text{CQ} \) over \( S \cup \text{sch}(\Sigma) \), and a tuple \( \bar{c} \in C^n \), the problem of deciding if \( \bar{c} \in \text{cert}_{q,\Sigma}(D) \) is undecidable [Beeri and Vardi, 1981]. This has led to a flurry of activity for identifying decidable syntactic restrictions. Such a restriction defines a subclass \( C \) of tgds, i.e., \( C \subseteq \text{TGD} \), which in turn gives rise to the decidable query language \((C, \text{CQ})\). The two main paradigms for ontological reasoning are guardedness and stickiness:

- **Guardedness:** A tgd is guarded if its body contains an atom, called guard, that contains all the body-variables [Calì et al., 2013]. Let \( G \) be the class of sets of guarded tgds. A key subclass of guarded tgds is the class of linear tgds, that is, tgds whose body consists of a single atom [Calì et al., 2012a]. Let \( L \) be the class of sets of linear tgds.

- **Stickiness:** The goal of stickiness is to express joins that are not expressible via guarded tgds. The key property underpinning this condition is as follows: During the chase, terms that are associated with variables that appear more than once in the body of a tgd (i.e., join variables) are always propagated (or “stick”) to the inferred atoms. Due to lack of space we omit the definition that can be found in [Calì et al., 2012b]. Let \( S \) be the class of sticky sets of tgds.

**Weak Versions:** Both \( G \) and \( S \) have a weak version: Weakly-guarded [Calì et al., 2013] and weakly-sticky [Calì et al., 2012b], respectively. These are highly expressive classes obtained by relaxing guardedness and stickiness so that only those positions that receive nulls during the chase are taken into account. We write \( \text{WG} \) and \( \text{WS} \) for the class of weakly-guarded and weakly-sticky sets of tgds, respectively.

We refer to the query languages defined above as ontology-mediated query languages.

### 3 Distribution of Queries and Connectedness

Our goal is to syntactically characterize the expressive power of the fragment of the query languages in question that guarantees distribution over components. Let us recall the notion of distribution over components [Ameloot et al., 2014]. A set of atoms \( A \) is connected if for all \( a, b \in \text{dom}(A) \), there exists a sequence \( \alpha_1, \ldots, \alpha_n \) of atoms in \( A \) such that \( a \in \text{dom}(\alpha_1), b \in \text{dom}(\alpha_n), \text{ and } \text{dom}(\alpha_i) \cap \text{dom}(\alpha_{i+1}) \neq \emptyset \), for each \( i \in [n-1] \).\(^3\) We call \( B \subseteq A \) a component of \( A \) if

\(^2\)For brevity, we use \( Q \) both for \((S, \Sigma, q)\) and the mapping \( q_Q \).

\(^3\)Henceforth, we write \( [k] \) for \( \{1, \ldots, k\} \).

\(^4\)For technical clarity, the notion of component is defined only for sets of atoms that do not contain nullary atoms.

\(^5\)We write \( \text{conUCQ} \) for the class of all queries definable by a union of connected CQs.
We consider each one of the above groups, and show that distribution over components implies (semantic) connectedness.

**Group A of Query Languages**

Our main technical tool is the so-called connecting operator, based on a construction given in [Ameloot et al., 2015] for showing that distribution over components implies connectedness when we focus on Datalog queries. By applying the connecting operator to a query \( Q = (S, \Sigma, q) \in (\text{TGD}, \text{CQ}) \), we obtain the query \( Q^c = (S, c(\Sigma), c(q)) \) defined as follows. The set \( c(\Sigma) \) consists of the tgds:

\[
R(x_1, \ldots, x_n) \rightarrow \text{con}(x_i, x_j),
\]

for each \( R/n \in S \) and \( i, j \in [n], \) where \( \text{con} \) is a new binary predicate not in \( \Sigma, \) and

\[
\text{con}(x, y) \rightarrow \text{con}(y, x),
\]

\[
\text{con}(x, y), \text{con}(y, z) \rightarrow \text{con}(x, z),
\]

stating that \( \text{con} \) is symmetric and transitive; \( \text{con}(a, b) \) states that the constants \( a \) and \( b \) belong to the same component of the input database. In addition, we have the tgds:

\[
\text{con}(x_1, v), R(x_1, \ldots, x_n) \rightarrow R^*(v, x_1, \ldots, x_n),
\]

where \( v \notin \{x_1, \ldots, x_m\} \), that annotates every atom of a certain component of the input database with all the constants in this component. Finally, for each \( \sigma \in \Sigma, \) we add to \( c(\Sigma) \) the connected version of \( \sigma \) obtained by replacing each atom \( R(\bar{x}) \) in \( \sigma \) with \( R^*(v_\sigma, \bar{x}) \), where \( v_\sigma \) is a new variable. Analogously, \( c(q) \) is defined as the connected version of \( q \). It can be shown that \( Q \in \text{DIST} \) implies \( Q \in Q^c \).

**Proposition 4** For each \( C \subseteq \text{TGD} \) that is closed under connecting, \( (C, \text{CQ}) \cap \text{DIST} \subseteq (\text{con}C, \text{conCQ}). \)

Interestingly, TGD, WG, and WS are closed under connecting. Hence, Proposition 4 implies that distribution over components implies connectedness for the languages of Group A.

**Group B of Query Languages**

It is clear that \( \text{G} \) and \( \text{L} \) are not closed under connecting, and therefore we need to follow a different approach. To this end, we exploit the fact that guarded and linear tgds are, by definition, connected. In particular, we establish a general result for subclasses of conTGD:

**Proposition 5** For each \( C \subseteq \text{conTGD}, \ (C, \text{CQ}) \cap \text{DIST} \subseteq (C, \text{conCQ}). \)

It is clear that the above result immediately shows that distribution over components implies connectedness for the languages of Group B. We proceed to explain the key ideas underlying Proposition 5. Fix an arbitrary class \( C \subseteq \text{conTGD}, \) and a query \( Q = (S, \Sigma, q) \in (C, \text{CQ}) \cap \text{DIST} \), where \( q/n \) is defined by the CQ \( \exists y \phi(x, y). \) Observe that if \( Q \) is unsatisfiable, i.e., there is no database \( D \) over \( S \) such that \( Q(D) \neq \emptyset \), then the claim holds trivially; simply choose an arbitrary unsatisfiable query from \( (C, \text{conTGD}). \) The interesting case is when \( Q \) is satisfiable. Let \( \{\phi_1, \ldots, \phi_k\} \) be the components of \( q. \) We assume that \( k \geq 2 \) since for \( k = 1 \) the claim holds trivially. Our goal is to show that there exists \( i \in [k] \) such that \( Q_i = (S, \Sigma, q_i), \) where \( q_i \) is the CQ obtained from \( q \) by keeping only the component \( \phi_i, \) is equivalent to \( Q. \) Since \( Q_i \in (C, \text{conCQ}) \) the claim follows. We proceed by distinguishing the two cases where \( q \) is either non-Boolean (i.e., \( n > 0 \)) or Boolean (i.e., \( n = 0 \)).

**Case 1 - Non-Boolean:** Interestingly, we can show that all the answer variables of \( q \) occur in a single component. In fact, if an answer variable occurs in more than one component of \( q, \) then \( Q \) is unsatisfiable, which contradicts the fact that \( Q \) is satisfiable; this exploits the fact that \( \Sigma \) is connected. Therefore, we can refer to the component \( q \) of \( q \) that contains the answer variables of \( q. \) Notably, the query obtained from \( Q \) by replacing \( q \) with \( q \) is equivalent to \( Q. \)

**Lemma 6** \( Q \equiv Q_z, \) where \( Q_z = (S, \Sigma, q_z). \)

It is clear that \( Q_z \in (C, \text{conCQ}), \) and the claim when \( q \) is non-Boolean follows by Lemma 6.

**Case 2 - Boolean:** Let us now consider the Boolean case. Although we cannot refer to the component \( q_z \) of \( q, \) we can show that there exists a component that gives rise to a query equivalent to \( Q. \) Given a query \( Q = (S, \Sigma, q) \in (C, \text{CQ}) \cap \text{DIST}, \) where \( q \) is Boolean and \( \psi_1, \ldots, \psi_m \) are its components, let \( Q_i \) be the query obtained from \( Q \) by keeping only the component \( \psi_i, \) and \( Q^{-}_i \) the query obtained from \( Q \) by removing the component \( \psi_i. \) We can show that, for each \( i \in [m], \) \( Q_i \subseteq Q^{-}_i \) or \( Q^{-}_i \subseteq Q_i. \) Thus, starting from \( Q, \) and repeatedly applying this result, we will find an integer \( i \in [k] \) such that \( Q_i \equiv Q^{-}_i \). It is not difficult to show that \( Q \equiv Q_i. \) Thus:

**Lemma 7** There exists \( i \in [k] \) such that \( Q \equiv Q_i, \) where \( Q_i = (S, \Sigma, q_i). \)

It is clear that the query \( Q_i \) provided by Lemma 7 belongs to \((C, \text{conCQ}), \) and the claim when \( q \) is Boolean follows.

**Description Logics.** Before we proceed to the third group of query languages, we would like to briefly discuss an interesting consequence of Proposition 5 to ontology-mediated query languages based on Description Logics (DLs). As for classes of tgds, a DL \( \mathcal{L} \) gives rise to the query language \( (\mathcal{L}, \text{CQ}). \) One of the most prominent and well-studied DLs is \( \mathcal{EL} \) [Baader, 2003]. It is known that an \( \mathcal{EL}-\text{Box} \) can be rewritten as a set of guarded tgds [Call et al., 2012a]. This fact, together with Propositions 5 and 6, allow us to show that:

**Corollary 8** \((\mathcal{EL}, \text{CQ}) \cap \text{DIST} = (\mathcal{EL}, \text{conCQ}). \)

**Group C of Query Languages**

\( S \) is not closed under connecting, and \( S \not\subseteq \text{conTGD}. \) Thus, we need to apply different techniques for establishing the desired result. Our main technical tool is as follows:

**Proposition 9** For each \( C \subseteq \text{TGD} \) such that

1. \( (C, \text{CQ}) \leq \text{UCQ}; \)
2. \( \text{conUCQ} \leq (\text{conC}, \text{conCQ}), \)

it holds that \( (C, \text{CQ}) \cap \text{DIST} \leq (\text{conC}, \text{conCQ}). \)

\(^{6}\)This means that, for each set \( \Sigma \in C, \ c(\Sigma) \in C. \)

\(^{7}\)As usual, \( Q_1 \subseteq Q_2 \) means that for every source database \( D, \)

\( Q_1(D) \subseteq Q_2(D). \)
To establish Proposition 9, it suffices to show the following lemma, which can be done by exploiting similar ideas as the ones developed above for proving Proposition 5:

**Lemma 10** UCQ ∩ DIST ⊆ conUCQ.

Having the above proposition in place, to establish the desired result for (S, CQ) it remains to show that (S, CQ) ⊈ UCQ and conUCQ ⊈ (conS, conCQ). The former is implicit in [Gottlob et al., 2014], where it is shown that evaluation for (S, CQ) is UCQ-rewritable. The latter is shown by exploiting the fact that every CQ of the form ∃y φ(x, y) can be converted into the sticky tgd φ(x, y) → ans(x, y); this tgd is sticky since all the body-variables are propagated to the head.

**Database Query Languages.** We conclude this section by discussing some interesting consequences to central database query languages, in particular CQ, UCQ, and NRDat. First, notice that Proposition 3 can be established for CQ and UCQ. This fact, together with Lemma 10, which holds also for CQ, implies that CQ ∩ DIST (UCQ ∩ DIST) has the same expressive power as conCQ (conUCQ). Regarding NRDat, we can show that the class of tgds that corresponds to non-recursive Datalog programs enjoys the two properties stated in Proposition 9. Hence, Propositions 3 and 9 imply that:

**Corollary 11** For each language Q ∈ {CQ, UCQ, NRDat}, Q ∩ DIST = conQ.

4 Deciding Distribution Over Components

The question that comes up is whether distribution over components for the query languages considered so far can be decided. This problem, dubbed Dist(Q) with Q being a query language, accepts as input a query Q ∈ Q, and asks whether Q ∈ DIST. Notice that Theorem 2 only says that Dist(Q) is equivalent to the containment problem for Q. Thus, for every C ⊆ TGD that captures Datalog programs, we immediately obtain that Dist(C, CQ) is undecidable. TGD, WG and WS are such classes, and hence:

**Theorem 12** For Q ∈ {(TGD, CQ), (WG, CQ), (WS, CQ)}, Dist(Q) is undecidable.

This negative result rules out weakly-guarded and weakly-sticky sets of tgds. What about their non-weak versions? Do they ensure the decidability of Dist? This will be the subject of the remainder of this section.

**Distribution and Guardedness**

We show that our problem is decidable if we focus on the answer-guarded fragment of (G, CQ). A CQ q is **answer-guarded** if it includes an atom that contains all the answer variables [Bárány et al., 2013]. Let AGCQ be the class of all queries definable by some answer-guarded CQ. Notice that AGCQ is a broad class of queries, which includes all Boolean CQs. Our goal is to show that Dist(G, AGCQ) is decidable.

Fix a query Q = (S, Σ, q) ∈ (G, AGCQ), where {q1, . . . , qk} are the components of q. Recall that Q is the query obtained from Q by keeping only the component qi of q. By exploiting Proposition 3, and Lemmas 6 and 7, we can show that:

**Lemma 13** Q ∈ DIST iff one of the following holds:

1. Q is unsatisfiable;
2. There exists i ∈ [k] such that Qi ∈ (G, AGCQ) and Q ⊈ Q

Checking whether Q is unsatisfiable can be reduced to the containment problem for (G, AGCQ). Thus, Lemma 13 provides an algorithm for Dist(G, AGCQ) under the assumption that Cont̄(G, AGCQ) is decidable. It remains to show that:

**Proposition 14** Cont(G, AGCQ) is decidable.

The above result is shown by a reduction to Cont(GDat), where GDat is the class of queries definable by some guarded Datalog query [Bárány et al., 2012]. Guarded Datalog requires an atom with an extensional predicate to contain all the variables in the head. Cont(GDat) is in 2EXPSPACE as it can be reduced to the satisfiability problem for guarded negation fixed point logic [Bárány et al., 2015]. Our reduction exploits a construction given in [Bárány et al., 2013] for rewriting a query of Q = (S, Σ, q) ∈ (G, AGCQ) into an equivalent Datalog query QDat = (π, Ans), where each rule of π has a guard atom (possibly with an intensional predicate) that contains all the head-variables. This construction assumes that q is answer-guarded; this is the reason why we focus on answer-guarded queries. From the above discussion, we get that:

**Theorem 15** Dist(G, AGCQ) is decidable.

The decision procedure underlying Theorem 15 shows that Dist(G, AGCQ) is elementary. However, its exact complexity is unknown. Moreover, the decidability of Dist(G, CQ), i.e., when the CQ is not answer-guarded, is still open.

**Distribution and Linearity**

It is easy to verify that the proof of Theorem 15 applies even if we focus on linear tgds; thus, Dist(L, AGCQ) is decidable. Interestingly, by exploiting the fact that evaluation for (L, CQ) is UCQ-rewritable [Gottlob et al., 2014], we can significantly strengthen the above result. We can show that:

**Theorem 16** Dist(L, CQ) is PSPACE-complete, and Π2P-complete if the arity of the schema is fixed.

Fix a query Q = (S, Σ, q) ∈ (L, CQ). By applying query rewriting techniques, we can construct Qr ∈ UCQ such that Q ≡ Qr [Gottlob et al., 2014]. Clearly, Q ∈ DIST iff Qr ∈ DIST. Hence, it would be quite beneficial to first understand the problem Dist(UCQ). One may be tempted to think that a UCQ distributes over components if each of its disjuncts does. It is easy to show that this is not the case; consider, e.g., the UCQ ∃x∃y (R(x) ∧ S(y)) ∨ ∃x R(x). The reason for this is the fact that some disjuncts do not distribute over components maybe subsumed by other ones that distribute over components. To formalize this we need an

---

---

---

---

8 All queries definable by some non-recursive Datalog query.

9 For brevity, we write Dist(C, CQ) instead of Dist(⟨C, CQ⟩).
auxiliary notation. For a CQ \( q(\bar{x}) = \exists y \left( \bigwedge_{i=1}^{n} R_i(\bar{x}, y) \right) \), let \( D[q] \) the database \( \{ R_i(\bar{c}(\bar{x}), c(y)) \mid i \in [n] \} \) obtained after replacing each variable \( v \) in \( q \) with a fresh constant \( c(v) \). Then:

**Lemma 17** For \( Q(\bar{x}) \in UCQ \), \( Q \in \text{DIST} \) iff for each \( q \in Q \), there exists \( D' \in \text{co}(D[q]) \) such that \( c(\bar{x}) \in Q(D') \).

Let \( Q = (S, \Sigma, q(\bar{x})) \in (L, CQ) \), and let \( Q_r(\bar{x}) \in UCQ \) be the rewriting obtained by applying the rewriting algorithm in [Gottlob et al., 2014]; recall that \( Q \equiv Q_r \). This algorithm constructs \( Q_r \) starting from \( q \) and exhaustively applying two steps: rewriting and minimization. The rewriting step resolves an atom in the disjunct \( q' \) under consideration using a linear tgd of \( \Sigma \), while the minimization step is responsible for unifying some atoms of \( q' \). Lemma 17 implies that \( Q \not\in \text{DIST} \) iff there exists \( q' \in Q_r \) such that, for every \( D' \in \text{co}(D[q']) \), \( c(\bar{x}) \in Q(D') \). This suggests the following decision procedure for the complemnet of \( \text{Dist}(L, CQ) \):

1. Construct a CQ \( q'(\bar{x}) \in Q_r \) by nondeterministically applying rewriting and minimization steps.
2. For each \( D' \in \text{co}(D) \), if \( c(\bar{x}) \in Q(D') \), then \textbf{Reject};
3. \textbf{Accept}.

Each step of the above procedure uses polynomial space. This is shown by exploiting the fact that each \( q' \in Q_r \) cannot have more atoms than the original CQ \( q \) due to the linearity of tgd, and also the fact that evaluation for \( (L, CQ) \) is in PSPACE; the latter is implicit in [Johnson and Klug, 1984]. In case the arity of the query is fixed, \( q' \) can be nondeterministically constructed in polynomial time, while evaluation for \( (L, CQ) \) is in NP: implicit in [Johnson and Klug, 1984]. Thus, the above procedure gives a \( \Sigma_2^p \) upper bound. Hence, \( \text{Dist}(L, CQ) \) is in PSPACE, and in \( \Pi_2^p \) if the arity of the schema is fixed.

The lower bounds are established by a reduction from a restricted version of \( \text{Cont}(L, CQ) \). We define \( \text{RestCont}(C, CQ) \) to be the problem of checking \( Q_1 \subseteq Q_2 \), where \( Q_1 = (S \cup \{ P \}, \Sigma, q_1) \) and \( Q_2 = (S \cup \{ P \}, \Sigma, q_2) \) with \( P \not\in S \) being a unary predicate not in \( \Sigma \), \( q_1, q_2 \), given: (i) a set \( \Sigma \in C \) of being connected tgd, (ii) a Boolean connected CQ \( q_1 \) with at least one variable \( v_1 \) that occurs only in atoms with a predicate \( S \), and (iii) a Boolean CQ \( q_2 \) that uses only predicates of \( S \). It holds that \( Q_1 \subseteq Q_2 \) iff \( (S \cup \{ P \}, \Sigma, q_1) \wedge P(v_1) \wedge q_2 \in \text{DIST} \), which in turn implies that we have a logspace reduction from \( \text{RestCont}(L, CQ) \) to \( \text{Dist}(L, CQ) \). Therefore, to obtain the desired lower bounds for \( \text{Dist}(L, CQ) \), it remains to show that \( \text{RestCont}(L, CQ) \) is PSPACE-hard, and \( \Pi_2^p \)-hard for fixed arity. The former can be shown by a reduction from query answering under linear tgd, while the latter is implicit in [Bienvenu et al., 2012] (see the proof of Theorem 9). In fact, the \( \Pi_2^p \)-hardness holds even for tgd of the form \( P(x) \rightarrow R(x) \).

**Distribution and Stickiness**

We proceed to show that \( \text{Dist}(S, CQ) \) is decidable, and pinpoint the complexity of the problem:

**Theorem 18** \( \text{Dist}(S, CQ) \) is co\(\text{NP}\)-complete, and \( \Pi_2^p \)-complete if the arity of the schema is fixed.

Evaluation for \( (S, CQ) \) is UCQ-rewritable [Gottlob et al., 2014]. This allows us to exploit the algorithm devised above for solving the complement of \( \text{Dist}(L, CQ) \). However, since sticky tgd may have more than one atom in their body, the space we need during the execution of the algorithm is not polynomial anymore; actually, we need exponential space. Nevertheless, it can be shown that for \( (S, CQ) \) this algorithm terminates after exponentially many steps, which implies a NE\(\text{XP}\)TIME upper bound for the complement of our problem. This exploits the fact that query evaluation for \( (S, CQ) \) is in \( \text{EXPTIME} \) [Calì et al., 2012b]. In case the arity of the schema is fixed, the same analysis as for \( (L, CQ) \) applies, and thus we obtain a \( \Sigma_3^p \) upper bound for the complement of our problem. The fact that evaluation for \( (S, CQ) \) in case of fixed arity is in NP has been shown in [Łukasiewicz et al., 2015].

To show co\(\text{NE}\)\(\text{XP}\)TIME-hardness for \( \text{Dist}(S, CQ) \), we first establish the same for \( \text{cont}(\text{conS}, \text{conCQ}) \) by giving a reduction from the so-called \textit{Exponential Tiling Problem} [Johnson, 1990]; for technical reasons, we assume databases with at least two constants. The desired co\(\text{NE}\)\(\text{XP}\)TIME-hardness is obtained by reducing \( \text{cont}(\text{conS}, \text{conCQ}) \) to \( \text{Dist}(S, \text{conCQ}) \). The \( \Pi_2^p \)-hardness in case the arity of the schema is fixed is obtained in the same way as for \( (L, CQ) \).

**Description Logics and Database Query Languages**

Interestingly, by exploiting the ideas developed above for understanding the complexity of \( \text{Dist} \), we can show that:

**Theorem 19** \( \text{Dist}(\Sigma, CQ) \) is \( \text{XP}\)TIME-complete.

For the upper bound, we relax Lemma 13 in order to deal with arbitrary (beyond answer-guarded) CQs, and then use the fact that \( \text{Cont}(\Sigma, CQ) \) is in \( \text{EXPTIME} \) [Bienvenu et al., 2012]. The lower bound exploits the problem \( \text{RestCont}^c \) obtained from \( \text{RestCont} \) by dropping the third condition in the definition of an instance of \( \text{RestCont} \). \( \text{RestCont}^c(\Sigma, CQ) \) is known to be \( \text{XP}\)TIME-hard [Bienvenu et al., 2012], which is not the case for \( \text{RestCont}(\Sigma, CQ) \).

We conclude with the complexity of \( \text{Dist} \) when we focus on central database query languages. We show that:

**Theorem 20** \( \text{Dist}(CQ) \) and \( \text{Dist}(UCQ) \) are \( \text{NP}\)-complete, even if the arity of the schema is fixed.

By exploiting Lemma 17, we can design a guess-and-check algorithm for \( \text{Dist}(UCQ) \) that runs in polynomial time; the \( \text{NP}\)-hardness is shown by reduction from \( \text{Cont}(CQ) \). Finally, by giving a proof similar to the one for Theorem 18, we show:

**Theorem 21** \( \text{Dist}(\text{NDat}) \) is co\(\text{NE}\)\(\text{XP}\)TIME-complete, even if the arity of the schema is fixed.\(^{11}\)

**5 Conclusions**

Here are some interesting open problems that we are planning to tackle: (i) The complexity of deciding distribution over components for guarded-based queries is still open, (ii) distribution over components in the presence of equality and denial constraints is not well understood, and (iii) we do not know how distribution over components behaves in the case of non-monotonic queries.

**Acknowledgements.** Austrian Science Fund (FWF), projects P25207-N23, Y698 and W1255-N23, and Vienna Science and Technology Fund (WWTF), project ICT12-015.
References


