Towards Universal Languages for Tractable Ontology Mediated Query Answering

Heng Zhang,[1] Yan Zhang,[2,3] Jia-Huai You,[4] Zhiyong Feng,[1] Guifei Jiang[5]*

[1]College of Intelligence and Computing, Tianjin University, Tianjin, China
[2]School of Computing, Engineering and Mathematics, Western Sydney University, Penrith, Australia
[3]School of Computer Science and Technology, Huazhong University of Technology and Science, Wuhan, China
[4]Department of Computing Science, University of Alberta, Edmonton, Canada
[5]College of Software, Nankai University, Tianjin, China
{heng.zhang, zyfeng}@tju.edu.cn, yan.zhang@westernsydney.edu.au, jyou@ualberta.ca, g.jiang@nankai.edu.cn

Abstract

An ontology language for ontology mediated query answering (OMQA-language) is universal for a family of OMQA-languages if it is the most expressive one among this family. In this paper, we focus on three families of tractable OMQA-languages, including first-order rewritable languages and languages whose data complexity of the query answering is in AC^0 or PTIME. On the negative side, we prove that there is, in general, no universal language for each of these families of languages. On the positive side, we propose a novel property, the locality, to approximate the first-order rewritability, and show that there exists a language of disjunctive embedded dependencies that is universal for the family of OMQA-languages with locality. All of these results apply to OMQA with query languages such as conjunctive queries, unions of conjunctive queries and acyclic conjunctive queries.

Introduction

Ontology mediated query answering (OMQA) is a paradigm that generalizes the traditional database querying by enriching the database with a domain ontology (Poggi et al. 2008). This paradigm has played an important role in the semantic web (Calvanese et al. 2007; Baader, Brandt, and Lutz 2005), data modelling (Berardi, Calvanese, and De Giacomo 2005), data exchange (Fagin et al. 2005) and data integration (Lenzerini 2002), and has recently emerged as one of the central issues in knowledge representation as well as in databases.

A long-term major topic for OMQA is to identify proper languages that specify ontologies. There have been a large number of ontology languages proposed for OMQA since the mid 2000s. For instance, in description logics, the DL-Lite family (Calvanese et al. 2007), $\mathcal{EL}$-family (Baader, Brandt, and Lutz 2005) and other variants have been proposed and extensively studied. More recently, the Datalog± family, a.k.a. existential rule languages, or dependencies in databases, have been rediscovered as promising languages for OMQA, see, e.g., (Baget et al. 2011; Calì, Gottlob, and Lukasiewicz 2012; Calì, Gottlob, and Pieris 2012). Most of these languages enjoy good computational properties such as the first-order rewritability or PTIME data complexity.

While all these languages are of their specific features and hence are useful in different applications, it is not realistic to implement OMQA-systems for all of them. So a natural question arises: Can we find the largest one (in the expressiveness) among the family of first-order rewritable (or PTIME-tractable) OMQA-languages? Let us call the largest language in the above sense a *universal language*. Clearly, it is of great theoretical and practical importance to identify the existence of universal language w.r.t. some kind of tractability, which is also the main task of this paper.

It is worth noting that the universality is one of the major principles for designing languages in both computer science and logic. For example, almost all the traditional programming languages, including C, Java and Prolog, are known to be universal for the family of Turing complete programming languages; propositional logic can express all boolean functions; and by the well-known Lindström theorem the first-order logic is the largest one among the logics that enjoy both the compactness and the Löwenheim-Skolem property; see, e.g., (Ebbinghaus, Flum, and Thomas 1994). In databases, first-order language is shown to be universal for the family of query languages with data complexity in AC^0, and Datalog universal for the family of query languages with data complexity in PTIME; see, e.g., (Immerman 1999).

Some recent work in OMQA has been done along the line of identifying universal languages. Calvanese et al. (2013) proved that, under a certain syntactic classification, some languages in the DL-Lite family are the maximal fragments of description logic with the first-order rewritability. By regarding OMQA as traditional database querying, Gottlob, Rudolph, and Simkus (2014) showed that weakly-guarded tuple-generating dependencies (TGDs) captures the class of EXPTIME queries; Rudolph and Thomazo (2015) proved that general TGDs captures the class of recursively enumerable queries. In the setting of schema mapping, Zhang, Zhang, and You (2015) showed that weakly-acyclic TGDs is the most expressive language of TGDs with finite semi-oblivious chase. All of these results shed new insights on understanding the expressiveness of existential rules, but it is worth noting that OMQA is significantly different from both traditional database querying and schema mapping. To understand the expressiveness in the framework of OMQA,

*Corresponding author.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Zhang, Zhang, and You (2016) proved that the language of disjunctive embedded dependencies is universal for the family of recursively enumerable OMQA-languages. Along this line, this paper will focus on tractable OMQA.

Aimed at exploiting universal languages for the tractable OMQA, in this paper we focus on three families of OMQA-languages, including first-order rewritable languages and languages whose data complexity is in AC^0 or PTIME. Our contributions are summarized as follows. On one hand, we prove that there is, in general, no universal language for each of the above families of languages. On the other hand, by restricting the number of database constants involved in query answering, we propose a novel property, called the *locality*, to approximate the first-order rewritability, and identify the existence of universal language for the family of local OMQA-languages. All of these results hold for OMQA with query languages such as conjunctive queries, unions of conjunctive queries and acyclic conjunctive queries.

Preliminaries

Databases and Instances. We use a countably infinite set of *constants* and a countably infinite set of *variables*, and assume they are disjoint. Every *term* is either a constant or a variable. A *relational schema* $\mathscr{R}$ consists of a set of *relation symbols*. Each relation symbol has an *arity* which is a natural number. An *atoms over* $\mathscr{R}$ (or $\mathscr{R}$-atom) is either an equality, or a *relational atom* built upon terms and relation symbols in $\mathscr{R}$. A *fact* is a variable-free relational atom. Each *instance over* $\mathscr{R}$ (or $\mathscr{R}$*-instance*) consists of a set of facts over $\mathscr{R}$. Instances that are finite are called *databases*. Suppose I is an instance. Let $adom(I)$ denote the set of constants that occur in I. Let DB$[\mathscr{R}]$ denote the class of all databases over schema $\mathscr{R}$. Given a set A of constants, by $I|_A$ we denote the subset of I in which each fact involves only constants in A.

Let I and J be instances over a relational schema $\mathscr{R}$, and $C \subseteq adom(I) \cap adom(J)$. Then every C*-homomorphism* from I to J is a function $h : adom(I) \to adom(J)$ such that (i) $R(\vec{a}) \in I$ implies $R(h(\vec{a})) \in J$ for all relation symbols $R \in \mathscr{R}$ and all tuples $\vec{a}$ of constants, and (ii) $h(c) = c$ for all $c \in C$. If such h exists, we say that I is C-homomorphic to J, and write $I \to_C J$; in addition, we write $I \twoheadrightarrow_C J$ if h is injective. For simplicity, C will be dropped if it is empty.

Queries. Fix $\mathscr{R}$ as a relational schema. By a *query* over $\mathscr{R}$ (or $\mathscr{R}$*-query*) we mean a formula built upon atoms over $\mathscr{R}$ in some logic. The logic could be first-order logic, second-order logic, or other variants. A query is *boolean* if it has no free variables. For convenience, given any query q, let $const(q)$ denote the set of constants that occur in q.

Every first-order formula is called a *first-order query*. A *conjunctive query (CQ)* is a query of the form $\exists \vec{y}.\varphi(\vec{x}, \vec{y})$ where φ is a finite but nonempty conjunction of relational atoms. Let q be a boolean CQ. We use $[q]$ to denote the database that consists of all the atoms that appear in q, where variables in atoms are regarded as special constants. The *Gaifman graph* of q is an undirected graph with each term in q as a vertex, and with each pair of distinct terms as an edge if they cooccur in some atom in q. A boolean CQ is called *acyclic* if its Gaifman graph is acyclic. A *union of conjunctive query (UCQ)* is a first-order formula built upon atoms by connectives $\wedge, \vee$ and quantifier $\exists$. Clearly, every UCQ is equivalent to a disjunction of CQs.

Every *Datalog*$^\neg$ *program* consists of a finite set of *rules* of the form $\forall \vec{x} \forall \vec{y}(\varphi(\vec{x}, \vec{y}) \to \alpha(\vec{x}))$, where α is a relational atom and φ is a finite conjunction of atoms or negated atoms; α and φ are called the *head* and the *body* of the rule, respectively. Each variable in $\vec{x}$ should have at least one positive occurrence in φ. A relation symbol is called *intentional* if it has at least one occurrence in the head of some rule, and *extensional* otherwise. No intensional relation symbol is allowed to appear in a negated atom. A *Datalog*$^\neg$ *query* is of the form $(\Pi, P)(\vec{x})$ where Π is a Datalog$^\neg$ program, P an extensional relation symbol, and $\vec{x}$ a variable tuple of a proper length. It is well-known that every Datalog$^\neg$ query can be translated to an equivalent formula in least fixpoint logic, see, e.g., (Ebbinghaus and Flum 1995).

Only boolean queries will be used in this work. For convenience, we employ CQ, ACQ and UCQ to denote the classes of boolean CQs, boolean acyclic CQs and boolean UCQs, respectively. Let FO denote the class of boolean first-order queries, FO$(+, \times)$ denote the class of boolean first-order queries that involve two built-in arithmetic relations $+$ and $\times$, and Datalog$^\neg(\leq)$ denote the class of boolean Datalog$^\neg$ queries that involve a built-in successor relation *Succ*, and special constants *min* and *max*, denoting the minimum and the maximum elements, respectively, under the underlying order. Given a class $\mathcal{C}$ of queries and a relational schema $\mathscr{R}$, let $\mathcal{C}[\mathscr{R}]$ denote the class of $\mathscr{R}$-queries that belong to $\mathcal{C}$.

In the theory of descriptive complexity (Immerman 1999), it was proved that FO$(+, \times)$ and Datalog$^\neg(\leq)$ exactly capture complexity classes AC^0 and PTIME respectively, where AC^0 denotes the class of languages recognized by a uniform family of circuits with constant depth and polynomial size, and PTIME denotes the class of languages recognized by a deterministic Turing machine in polynomial time.

Dependencies. A *disjunctive embedded dependency (DED)* over a relational schema $\mathscr{R}$ is a sentence σ of the form

$$\forall \vec{x} \forall \vec{y}(\phi(\vec{x}, \vec{y}) \to \exists \vec{z}_1.\psi_1(\vec{x}, \vec{z}_1) \vee \cdots \vee \exists \vec{z}_k.\psi_k(\vec{x}, \vec{z}_k))$$

where $k \geq 0$, ϕ is a conjunction of relational $\mathscr{R}$-atoms involving terms from $\vec{x} \cup \vec{y}$ only, each ψ_i is a conjunction of atoms over $\mathscr{R}$ involving terms from $\vec{x} \cup \vec{z}_i$ only, and each variable in $\vec{x}$ has at least one occurrence in ϕ. In particular, σ is called a *tuple-generating dependency (TGD)* if it is equality-free and $k = 1$. For simplicity, we omit the universal quantifiers and the brackets appearing outside the atoms.

Let D be a database, Σ a set of (first-order) sentences, and q a boolean query; all of them are over a common relational schema $\mathscr{R}$. We write $D \cup \Sigma \models q$ if, for all $\mathscr{R}$-instances I with $D \subseteq I$, if $I \models \sigma$ for all sentences $\sigma \in \Sigma$ then $I \models q$, where the satisfaction relation $\models$ is defined in the standard way.

Ontologies and Languages in OMQA

Before identifying the existence of universal languages for OMQA, we need some notions to clarify what an ontology in OMQA is, and what an ontology language in OMQA is. To make the presented results more applicable, we will define these notions in a language-independent way.

To define ontologies in OMQA, below we generalize the notion introduced in (Zhang, Zhang, and You 2016) from CQs to more general query languages such as UCQs.

Definition 1. Let $\mathscr{D}$ and $\mathscr{Q}$ be relational schemas, and $\mathcal{Q}$ a class of queries. A *quasi-OMQA[$\mathcal{Q}$]-ontology* over $(\mathscr{D}, \mathscr{Q})$ is a set of ordered pairs (D, q), where D is a $\mathscr{D}$-database and q a boolean $\mathscr{Q}$-query in $\mathcal{Q}$ such that $\mathit{const}(q) \subseteq \mathit{adom}(D)$.

Moreover, a quasi-OMQA[$\mathcal{Q}$]-ontology O over $(\mathscr{D}, \mathscr{Q})$ is called an *OMQA[$\mathcal{Q}$]-ontology* if all of the following hold:

1. O is *closed under query conjunctions*, i.e., $(D, q) \in O \,\&\, (D, p) \in O \,\&\, q \wedge p \in \mathcal{Q} \Longrightarrow (D, q \wedge p) \in O$.
2. O is *closed under query implications*, i.e., $(D, q) \in O \,\&\, p \in \mathcal{Q} \,\&\, q \models p \Longrightarrow (D, p) \in O$.
3. O is *closedunderinjectivedatabasehomomorphisms*,i.e., $(D, q) \in O \,\&\, D \twoheadrightarrow_{\mathit{const}(q)} D' \Longrightarrow (D', q) \in O$.

Given any logical theory Σ, we can interpret it as a quasi-OMQA[$\mathcal{Q}$]-ontology over $(\mathscr{D}, \mathscr{Q})$ as follows:

$$[\![\Sigma]\!]^{\mathcal{Q}}_{\mathscr{D},\mathscr{Q}} = \{(D, q) : D \in \mathrm{DB}[\mathscr{D}] \,\&\, q \in \mathcal{Q}[\mathscr{Q}] \,\&\, D \cup \Sigma \models q\}.$$

It is easy to see that, for theories Σ in almost all the classical logic, $[\![\Sigma]\!]^{\mathcal{Q}}_{\mathscr{D},\mathscr{Q}}$ is indeed an OMQA[$\mathcal{Q}$]-ontology.

With the notion of ontology, we are then able to present an abstract definition for ontology languages in OMQA.

Definition 2. Let V be a finite but nonempty set, $\mathscr{D}$ and $\mathscr{Q}$ relational schemas, and $\mathcal{Q}$ a class of queries. Then every *OMQA[$\mathcal{Q}$]-language* $\mathcal{L}$ over $(\mathscr{D}, \mathscr{Q})$ (with vocabulary V) is defined as an ordered pair (T, M) such that:

1. T consists of a decidable set of *theories*, each of which is a finite string over V (i.e., an element of V^*);
2. M is a *semantic mapping*, i.e., a function that maps each theory in T to an OMQA[$\mathcal{Q}$]-ontology over $(\mathscr{D}, \mathscr{Q})$.

Example 1. Let $\mathscr{D}$ and $\mathscr{Q}$ be relational schemas, $\mathcal{Q}$ a class of queries, and T a decidable class of finite sets of DEDs. Let M be a function that maps each set $\Sigma \in T$ to $[\![\Sigma]\!]^{\mathcal{Q}}_{\mathscr{D},\mathscr{Q}}$. It is easy to see that $\mathcal{L} = (T, M)$ is an OMQA[$\mathcal{Q}$]-language.

The language $\mathcal{L}$ defined above is called a *DED-language* over $(\mathscr{D}, \mathscr{Q})$ (induced by T). In particular, if T consists of all finite sets of DEDs, we call it the *full DED-language* over $(\mathscr{D}, \mathscr{Q})$. Unfortunately, it had been proved in (Vardi 1982) that query answering with the full DED-language is uncomputable. In this work, we thus focus on tractable OMQA-languages. We will consider two kinds of tractability:

Definition 3. Let $\mathscr{D}$ and $\mathscr{Q}$ be relational schemas, $\mathcal{C}$ and $\mathcal{Q}$ classes of queries, and $\mathcal{K}$ a complexity class. An OMQA[$\mathcal{Q}$]-language $\mathcal{L} = (T, M)$ over $(\mathscr{D}, \mathscr{Q})$ is

1. *$\mathcal{C}$-rewritable* if there is a computable function rew that maps each ordered pair $(t, q) \in T \times \mathcal{Q}[\mathscr{Q}]$ to a boolean query $\varphi_{t,q} \in \mathcal{C}[\mathscr{D}]$ such that $(D, q) \in M(t)$ iff $D \models \varphi_{t,q}$; in this case, rew is called a *$\mathcal{C}$-rewriting function* of $\mathcal{L}$.
2. *$\mathcal{K}$-compilable* if there is a computable function com that maps each ordered pair $(t, q) \in T \times \mathcal{Q}[\mathscr{Q}]$ to a Turing machine $\mathbb{M}_{t,q}$, whose running time belongs to the class $\mathcal{K}$, such that $(D, q) \in M(t)$ iff $\mathbb{M}_{t,q}$ accepts on the input D; in this case, com is called a *$\mathcal{K}$-compiler* of $\mathcal{L}$.

Example 2. According to (Calì, Gottlob, and Lukasiewicz 2012), the language of linear TGDs is both FO-rewritable and AC^0-compilable, and the language of guarded TGDs is both Datalog$^\neg(\leq)$-rewritable and PTIME-compilable.

Remark 1. Clearly, there is a nonuniform way to redefine notions in Definition 3 by allowing rewriting functions and compilers to be uncomputable. However, it is worth noting that languages defined in such a way could be intractable. In fact, there is a nonuniform FO-rewritable OMQA-language in which the query answering is highly undecidable.

Next we give the definition of universal OMQA-language.

Definition 4. Let $\mathcal{Q}$ be a class of queries, $\mathscr{D}$ and $\mathscr{Q}$ relational schemas, and $\mathcal{L} = (T, M)$ and $\mathcal{L}' = (T', M')$ OMQA[$\mathcal{Q}$]-languages over $(\mathscr{D}, \mathscr{Q})$. Then we say that $\mathcal{L}'$ is *at least as expressive as* $\mathcal{L}$, written $\mathcal{L} \leq \mathcal{L}'$, if for each theory $t \in T$ there is a theory $t' \in T'$ such that $M(t) = M'(t)$; and $\mathcal{L}'$ *has the same expressiveness as* $\mathcal{L}$ if both $\mathcal{L} \leq \mathcal{L}'$ and $\mathcal{L}' \leq \mathcal{L}$.

An OMQA[$\mathcal{Q}$]-language $\mathcal{L}$ is called *universal* for a family $\mathscr{L}$ of OMQA[$\mathcal{Q}$]-languages over $(\mathscr{D}, \mathscr{Q})$ if (i) $\mathcal{L} \in \mathscr{L}$, and (ii) for all languages $\mathcal{L}' \in \mathscr{L}$, we have that $\mathcal{L}' \leq \mathcal{L}$.

Nonexistence for the General Case

One ambitious goal in OMQA is to find some universal language for the tractable OMQA. Unfortunately, the following theorem shows that this goal is in general unachievable.

Theorem 1. *Let $\mathscr{D}$ and $\mathscr{Q}$ be relational schemas such that $\mathscr{Q}$ contains a relation symbol of arity ≥ 2, and suppose $\mathcal{C} \in \{FO, FO(+,\times), Datalog^\neg(\leq)\}$ and $ACQ \subseteq \mathcal{Q} \subseteq UCQ$. Then there is no universal language for the family of $\mathcal{C}$-rewritable OMQA[$\mathcal{Q}$]-languages over $(\mathscr{D}, \mathscr{Q})$.*

Since AC^0 and PTIME are exactly captured by FO$(+,\times)$ and Datalog$^\neg(\leq)$ respectively, by Theorem 1 we have

Corollary 2. *Let $\mathscr{D}$ and $\mathscr{Q}$ be relational schemas such that $\mathscr{Q}$ contains at least one relation symbol of arity ≥ 2, and suppose $\mathcal{K} \in \{AC^0, PTIME\}$ and $ACQ \subseteq \mathcal{Q} \subseteq UCQ$. Then there is no universal language for the family of $\mathcal{K}$-compilable OMQA[$\mathcal{Q}$]-languages over $(\mathscr{D}, \mathscr{Q})$.*

To prove Theorem 1, the general idea is to implement a diagonalization argument as follows. Assume by contradiction that there is a universal language for the desired family. We first give an effective enumeration for all nontrivial ontologies defined in the universal language. With this enumeration, we then construct a new OMQA[$\mathcal{Q}$]-ontology O and a new language $\mathcal{L}'$ in which O is definable; Finally we show that $\mathcal{L}'$ is still $\mathcal{C}$-rewritable, which leads to a contradiction.

Proof of Theorem 1. Only consider the case where $\mathcal{C} =$ FO and $\mathcal{Q} =$ UCQ. Assume by contradiction that there is a universal language for FO-rewritable OMQA[UCQ]-languages over $(\mathscr{D}, \mathscr{Q})$. Let $\mathcal{L} = (T, M)$ be such a language. Our task is to define another FO-rewritable OMQA[UCQ]-language that is strictly more expressive than $\mathcal{L}$. To do this, we first construct an ontology that is not definable in $\mathcal{L}$.

Before we present the construction, some notations are needed. W.l.o.g., we assume that there is a binary relation symbol R in $\mathscr{Q}$. Note that, by a repetition of the parameters, R can be always simulated by another relation symbol of arity > 2. For example, one can use $S(x, x, y)$ to simulate $R(x, y)$. With this assumption, we first define a sequence of acyclic CQs. For all integers $n \geq 1$, we define

$$q_n = \exists x_0 \cdots \exists x_n (R(x_0, x_1) \wedge \cdots \wedge R(x_{n-1}, x_n)).$$

Intuitively, q_n asserts that there is a cycle-free path (via R) of length $n + 1$ in the intended model.

Let $\vec{s} = (s_1, s_2, \dots)$ be an effective enumeration[1] of all the theories in T. Such an enumeration clearly exists. Now our task is to construct countably infinite sequences $\vec{N}$ and $\vec{t}$, where $\vec{N} = (N_1, N_2, \dots)$ is a sequence of positive integers, and $\vec{t} = (t_1, t_2, \dots)$ is a sequence of theories in T. The sequences are required to have the following properties:

1. $\vec{N}$ is monotonic increasing, i.e., $N_i \leq N_j$ if $i < j$;
2. For all $k \geq 1$ there exists a database D with $(D, q_k) \in M(t_k)$ and $|adom(D)| \leq N_k$;
3. For all $t \in T \backslash \vec{t}$ there exists $i > 0$ such that $|adom(D)| > N_i + 1$ for all databases D with $(D, q_i) \in M(t)$.

Procedure 1 is devoted to generate the desired sequences.

Procedure 1: Generating Sequences $\vec{t}$ and $\vec{N}$

```
1  $n \leftarrow 1$ ;
2  for $i \leftarrow 1$ to $\infty$ do
3      for $n \leftarrow n$ to $\infty$ do
4          for $j \leftarrow 1$ to $n$ do
5              if $\exists D : (D, q_i) \in M(s_j)$ & $|adom(D)| \leq n$
                 then goto line 9 ;
6              if $\exists D : (D, q_i) \in M(s_j)$ & $|adom(D)| \leq$
                 $n + 1$ then
7                  $n \leftarrow n + 1$ ;
8                  goto line 9 ;
9      $N_i \longleftarrow n$ ;
10     $t_i \longleftarrow s_j$ ;
11     delete $s_j$ from $\vec{s}$ ;
```

Now we have the following property:

Claim 1. The sequences $\vec{N}$ and $\vec{t}$ generated by Procedure 1 satisfy Properties (1-3).

Now we are able to construct the desired ontology. To do this, we first define some notations. For $n \geq 1$, let λ_n denote the sentence $\exists x_1 \cdots x_n \bigwedge_{1 \leq i < j \leq n} \neg(x_i = x_j)$, which asserts that the intended domain contains at least n elements. Given a boolean UCQ q, if there exists an integer $k \geq 1$ such that $q_k \models q$, let φ_q denote $\lambda_{N_m + 1}$ where m is the least integer among such ks, and let φ_q denote the sentence $\exists x \neg(x = x)$ (always false) if no such ks exist. Furthermore, we define

$$O = \{(D, q) : D \in \text{DB}[\mathscr{D}] \,\&\, q \in \text{UCQ}[\mathscr{Q}] \,\&\, D \models \varphi_q\}\,.$$

It is not difficult to prove the following properties:

Claim 2. O is an OMQA[UCQ]-ontology.

Claim 3. $O \neq M(t)$ for any theory $t \in T$.

With Claims 2 and 3, we are now in the position to prove the desired theorem. Let t' be a binary string such that $t' \notin T$, and let $T' = T \cup \{t'\}$. Following the decidability of T, we have the decidability of T'. Let M' be a function that extends M by mapping t' to O, and let $\mathcal{L}' = (T', M')$. By Claim 2, we know that $\mathcal{L}'$ is an OMQA[UCQ]-language. Suppose rew is an FO-rewriting function. Let rew' be a function that extends rew by mapping (t', q) to φ_q for all boolean UCQs q. By a slight modification to Procedure 1, one can easily devise an algorithm to compute N_i (and s_i) on given integer $i \geq 1$. This implies that rew' is computable. By definition, we know that rew is an FO-rewriting function, which implies that $\mathcal{L}'$ is FO-rewritable. By Claim 3, we also know that $\mathcal{L}'$ is strictly more expressive than $\mathcal{L}$, a contradiction as desired. And this completes the proof. □

Remark 2. Since the sentence φ_q defined in the above proof is also an FO$(+, \times)$-sentence, so the proof directly applies to the case of FO$(+, \times)$. For a proof of the remaining case, one can convert φ_q to a Datalog$^\neg(\leq)$-program.

Locality to the Rescue

In the last section, we proved that there is no universal language for tractable OMQA in general. Then, a natural question arises as to whether one can find a natural property that approximates the tractability but still allows the existence of a universal language. The challenge here is that the property should be manageable enough to avoid a diagonalization argument (see the proof of Theorem 1). Below we propose a property as an approximation of FO-rewritability.

Locality as Approximation of FO-rewritability

A *bound function* is a computable function $\ell : \mathbb{N} \to \mathbb{N}$ such that $\ell(n) \geq n$ for $n \in \mathbb{N}$. To simplify the presentation, we fix a way to represent bound functions, e.g., one can represent each bound function by a Turing machine that computes it. A class of bound functions is called *decidable* if the class of representations of those bound functions is decidable.

To measure the size of a query, we fix a function $||\cdot||$ that maps each UCQ to a nonnegative integer. Clearly, there are a number of methods to define $||\cdot||$. The only restriction here is that we require $||p \wedge q|| \geq ||p|| + ||q||$ for all UCQs p and q.

Definition 5. Let $\mathscr{D}$ and $\mathscr{Q}$ be relational schemas, and $\mathcal{Q}$ a class of queries, O an OMQA[$\mathcal{Q}$]-ontology over $(\mathscr{D}, \mathscr{Q})$, and ℓ a bound function. Then O is called *ℓ-local* if for all boolean $\mathscr{Q}$-queries $q \in \mathcal{Q}$ and all $\mathscr{D}$-databases D there is a set A, which consists of at most $\ell(||q||)$ constants, such that

$$(D, q) \in O \quad \text{iff} \quad (D|_A, q) \in O.$$

[1] I.e., there is a Turing machine to generate such an enumeration.

Furthermore, given an OMQA[$\mathcal{Q}$]-language $\mathcal{L}$, a bound function ℓ and a class F of bound functions, $\mathcal{L}$ is called *ℓ-local* if all OMQA[$\mathcal{Q}$] ontologies defined in $\mathcal{L}$ is ℓ-local, and $\mathcal{L}$ is F-local if it is ℓ'-local for some bound function $\ell' \in \mathsf{F}$.

One might question why the bounded locality is a good approximation to the first-order rewritability. Let $\exists^+$FO($\neq$) denote the class of first-order sentences built on atoms and inequalities by using connectives $\wedge, \vee$ and the quantifier $\exists$. Obviously, this class is exactly the class of UCQs with inequalities. It had been observed by Benedikt et al. that $\exists^+$FO($\neq$) captures the class of first-order sentences that preserved under injective homomorphisms (2016). It remains open whether such a preservation theorem holds on finite structures (or databases). If this is indeed true, by the following proposition we then have that an OMQA-language is FO-rewritable iff it is ℓ-local for some bound function ℓ.

Proposition 3. *Let $\mathscr{D}$ and $\mathscr{Q}$ be relational schemas, O an OMQA[UCQ]-ontology over $(\mathscr{D}, \mathscr{Q})$, and ℓ a bound function. Then O is ℓ-local iff for each boolean $\mathscr{Q}$-UCQ q there is a $\exists^+$FO($\neq$)-sentence φ with at most $\ell(||q||)$ quantifiers such that $(D, q) \in O$ iff $D \models \varphi$ for all $\mathscr{D}$-databases D.*

Remark 3. Proposition 3 reveals an intrinsic connection between the bounded locality and the complexity of rewritings. We will elaborate this in an extended version of this paper.

Universal Language for Local OMQA

Now it remains to know whether the bounded locality allows the existence of universal languages. For convenience, in the rest of this section, we *fix F as a decidable class of bound functions; fix $\mathscr{D}$ and $\mathscr{Q}$ as a pair of disjoint relational schemas*. The disjointness will not introduce any real limitation. For instance, in a DED-language, given any set Σ of DEDs, one can construct another set Σ' of DEDs by introducing a fresh relation symbol R' for each $R \in \mathscr{D}$, and adding copy rules of the form $R'(\vec{x}) \rightarrow R(\vec{x})$. Clearly, Σ' has the same behaviour over $(\mathscr{D}', \mathscr{Q})$ as Σ over $(\mathscr{D}, \mathscr{Q})$, where $\mathscr{D}'$ denotes the schema consisting of all the fresh symbols.

Surprisingly, we have the following result.

Theorem 4. *Let $\mathcal{Q}$ be a decidable class of UCQs. Then there exists a DED-language that is universal for the family of F-local OMQA[$\mathcal{Q}$]-languages over $(\mathscr{D}, \mathscr{Q})$.*

Let ℓ be any bound function in F. To prove Theorem 4, the general idea is to develop a transformation that converts every DED set to an ℓ-local DED set. In addition, for each DED set that is already ℓ-local, the transformation is required to preserve the semantics of query answering. If such a transformation exists, since DED is universal for the family of OMQA-languages in which query answering is recursively enumerable, we then obtain a universal language for the family of F-local OMQA-languages.

Let us begin with a finite set Σ of DEDs over a relational schema $\mathscr{R} \supseteq \mathscr{D} \cup \mathscr{Q}$. To implement the desired transformation, we first show how to construct an ℓ-local OMQA[$\mathcal{Q}$]-ontology from Σ. As a natural idea, one may expect to define the desired ontology by removing all the pairs (D, q) from the original ontology (defined by Σ) where q is not ℓ-local on D. Unfortunately, the ontology defined above is in general not well-defined. To construct the desired ontology, the ℓ-locality and the closure under both query conjunctions and query implications should be maintained simultaneously.

Below we explain how to construct the ontology. We need to fix a strict linear order $\prec$ over $\mathscr{Q}$-UCQs firstly. The strict linear order is required to satisfy $p \prec q$ for all $\mathscr{Q}$-UCQs p and q such that $||p|| < ||q||$. Clearly, such an order always exists. For the given set Σ of DEDs, let S_Σ be the set that consists of the ordered pair (D, q) if D is a $\mathscr{D}$-database, q is a $\mathscr{Q}$-UCQ in $\mathcal{Q}$, and the following condition holds:

$$\begin{aligned} &\forall p \in \mathcal{Q}[\mathscr{Q}] : ||p|| \leq ||\widehat{pr}(q)|| \,\&\, \widehat{pr}(q) \models p \\ &\quad \Longrightarrow \exists A \subseteq adom(D) \text{ s.t. } |A| \leq \ell(||p||) \,\&\, D|_A \cup \Sigma \models p \end{aligned} \tag{1}$$

where $pr(q)$ is the set of boolean UCQs p such that $p \prec q$ and $(D, p) \in S_\Sigma$, and $\widehat{pr}(q)$ denotes the conjunction of q and all UCQs in $pr(q)$. Moreover, we define O_Σ as the minimum superset of S_Σ that is closed under query conjunctions, query implications and injective database homomorphisms.

The constructed ontology enjoys several properties which will play important roles in our proof for Theorem 4.

Lemma 5. *If $[\![\Sigma]\!]^{\mathcal{Q}}_{\mathscr{D},\mathscr{Q}}$ is ℓ-local, then $O_\Sigma = [\![\Sigma]\!]^{\mathcal{Q}}_{\mathscr{D},\mathscr{Q}}$.*

Lemma 6. *O_Σ is an ℓ-local OMQA[$\mathcal{Q}$]-ontology.*

Proof. We first claim that, for all UCQs q with $(D, q) \in O_\Sigma$, there exists an integer $k \geq 1$ and UCQs $p_1, \ldots, p_k$ such that $(D, p_i) \in S_\Sigma$ for each p_i and $p_1 \wedge \cdots \wedge p_k \models q$. This can be proved by a routine induction of the construction of O_Σ.

With the claim, w.l.o.g., we assume $p_1 \prec p_2 \prec \cdots \prec p_k$. Let p denote the query $p_1 \wedge \cdots \wedge p_k$. Obviously, it holds that $\widehat{pr}(p_k) \models p$, which implies that $\widehat{pr}(p_k) \models q$ immediately. On the other hand, by definition we know that p_i is ℓ-local on D, i.e., there is a set $A_i \subseteq adom(D)$ such that $D|_{A_i} \cup \Sigma \models p_i$. Let $A = A_1 \cup \cdots \cup A_k$. We then have $D|_A \cup \Sigma \models p$ and $|A| \leq \ell(||p_1||) + \cdots + \ell(||p_k||) \leq \ell(||p_1|| + \cdots + ||p_k||) \leq \ell(||p||)$. Consequently, we obtain that p is ℓ-local on D.

Now, it remains to show that q is ℓ-local on D. For the case where $||p|| \leq ||q||$, from $p \models q$ and $D|_A \cup \Sigma \models p$, we obtain that $D|_A \cup \Sigma \models q$, which implies that q is ℓ-local on D. For the other case, it must be true that $||p|| > ||q||$. From the fact that $||\widehat{pr}(p_k)|| \geq ||p||$, we know that $||\widehat{pr}(p_k)|| > ||q||$. Since $(D, p_k) \in S_k$ is true, by Condition (1) we know that there is a set $B \subseteq adom(D)$ such that $D|_B \cup \Sigma \models q$, which means that q is ℓ-local on D. This then completes the proof. □

Lemma 7. *O_Σ is recursively enumerable.*

Now, to define the transformation, it remains to show how to encode the ontology O_Σ by another set of DEDs. Suppose D is the underlying database, and q the underlying query. The encoding will be implemented in the following way:

1. Simulate the query answering of q under O_Σ and D.
2. If the answer of Stage 1 is positive, then nondeterministically copy disjuncts of q to generate the universal models.

The main challenges of implementing the above encoding are as follows. Firstly, instead of a single universal model, we need to generate a set of universal models in Stage 2. It is not clear whether the technique of generating universal model in (Zhang, Zhang, and You 2016) can be applied to this situation. Secondly, to encode the computation in Stage

1, a successor relation is needed. But it seems impossible to define such a relation in the language of DEDs explicitly.

Below we explain how to implement the encoding.

Defining Successor and Arithmetic Relations. To implement the desired encoding, a successor relation needs to be defined so that the constants in the underlying database D can be ranged over. As there is no negation in the body of DEDs, it seems impossible to construct DEDs to traverse ALL constants in $adom(D)$. Fortunately, thanks to the closure of OMQA-ontologies under injective database homomorphisms, we do not need a successor relation on the full domain. The reason is as follows. Suppose we want to show that a query q is derivable from the database D under some ontology O. As O is closed under injective database homomorphisms, it is equivalent to show *whether there is a subset* A *of* $adom(D)$ *such that* q *is derivable from* $D|_A$ *under* O.

To range over subsets of $adom(D)$, we employ the *partial successor relations* on $adom(D)$, each of which is a successor relation on some subset of $adom(D)$. Clearly, there is a partial successor relation for each subset A of $adom(D)$. With the mentioned property, we will define some DEDs to generate partial successor relations on $adom(D)$. To check whether q is derivable from D under O, it would be sufficient to test whether the computation of Stage 1 halts with "accept" under a certain partial successor relation.

Our method to generate partial successor relations was inspired by Rudolph and Thomazo's technique to define successor relations in the language of TGDs (2015). In that paper they showed that every homomorphism-closed database query can be defined by a set of TGDs. It is worth noting that the ontology mediated queries focused on this paper are not necessary to be closed under homomorphisms. So their technique cannot be applied directly. Fortunately, a linear order on $adom(D)$ can be easily defined by a set of DEDs, and with this order, we are able to use their idea to generate all partial successor relations that are compatible with the defined order. Now we show how to implement this idea.

Let *AD* be a unary relation symbol that will be interpreted as $adom(D)$. Clearly, such a relation can be easily defined by some DEDs. With the relation *AD*, a linear order relation *Less* over *AD* can then be defined in a routine way:

$$AD(x) \wedge AD(y) \rightarrow Less(x,y) \vee x = y \vee Less(y,x) \tag{2}$$

$$Less(x,y) \wedge Less(y,z) \rightarrow Less(x,z) \tag{3}$$

$$Less(x,x) \rightarrow \bot \tag{4}$$

To generate all partial successor relations compatible with *Less*, we link each constant c in $adom(D)$ with a *alias* a by the relation $Link(c,a)$. Suppose a_1 and a_2 are aliases of constants c_1 and c_2 respectively, by $Next(a_1,a_2)$ we mean that c_2 is the immediate successor of c_1 in the underlying successor relation. The head (resp., the tail) of a partial successor relation is denoted by $First(a)$ (resp., $Last(b)$). In particular, we use a as the *name* of the underlying relation. Every partial successor relation is required to have a head and a tail. To generate these relations, we use the following DEDs:

$$AD(x) \rightarrow \exists v\, Link(x,v) \wedge Last(v) \wedge First(v) \tag{5}$$

$$AD(x) \rightarrow \exists v\, Link(x,v) \wedge Last(v) \wedge Partial(v) \tag{6}$$

$$\begin{aligned} &Less(x,y) \wedge Link(y,v) \wedge Partial(v) \\ &\quad \rightarrow \exists u\, Link(x,u) \wedge Next(u,v) \wedge First(u) \end{aligned} \tag{7}$$

$$\begin{aligned} &Less(x,y) \wedge Link(y,v) \wedge Partial(v) \\ &\quad \rightarrow \exists u\, Link(x,u) \wedge Next(u,v) \wedge Partial(u) \end{aligned} \tag{8}$$

To understand how these DEDs work in more detail, please refer to the following example.

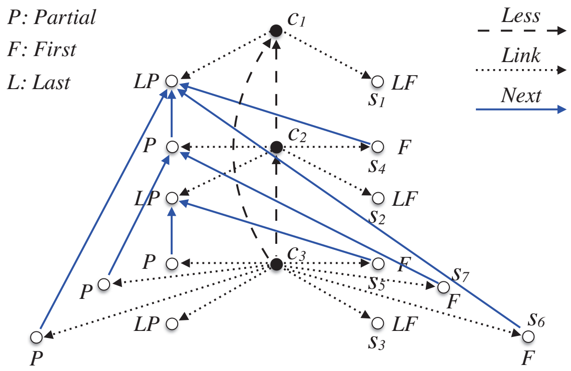

Figure 1: The Instance Generated in Example 3.

Example 3. Let D be a database which involves only constants c_1, c_2 and c_3, and suppose the linear order defined by *Less* is $c_1 > c_2 > c_3$. By an exhaustive application of DEDs (6-8),[2] we obtain an instance as illustrated by Figure 1.

As seen in Figure 1, there are 7 partial successor relations $s_1, \ldots, s_7$ generated in the instance. For instance, s_1 defines the partial successor relation involving only c_1; s_5 defines the relation $c_2 > c_3$; s_7 defines the relation $c_1 > c_2 > c_3$.

To encode Stage 1 mentioned before, we need to generate a linear order and the corresponding successor relation on a countably infinite domain, making sure they are compatible with the underlying partial successor relation on $adom(D)$. More relations are needed to do this. $Zero(o,a)$ means that a is the least element under the order o; $DMax(o,a)$ states that a is the largest element in $adom(D)$ under the order o; $Succ(o,a,b)$ denotes that b is the immediate successor of a under the order o; and $LT(o,a,b)$ asserts that a is less than b under the order o. For a technical reason, we also need some auxiliary relations. $TC(o,a)$ denotes that a is not less than the largest element in $adom(D)$ under the order o, and $RT(o,a,c)$ means that a is an alias of c and it is used to build

[2] To make the figure simple, we use the semi-oblivious chase.

the order o. All of these are defined by the following DEDs:

$$\mathit{First}(w) \wedge \mathit{Link}(x, w) \rightarrow \mathit{Zero}(w, x) \wedge \mathit{RT}(w, w, x) \tag{9}$$

$$\begin{aligned}\mathit{Next}(u, v) \wedge \mathit{RT}(w, u, x) \wedge \mathit{Link}(y, v)& \\ \rightarrow \mathit{Succ}(w, x, y) \wedge \mathit{RT}(w, v, y)&\end{aligned} \tag{10}$$

$$\mathit{Last}(v) \wedge \mathit{RT}(w, v, x)) \rightarrow \mathit{DMax}(w, x) \wedge \mathit{TC}(w, x) \tag{11}$$

$$\mathit{TC}(v, x) \rightarrow \exists z.\mathit{Succ}(v, x, y) \wedge \mathit{TC}(v, y) \tag{12}$$

$$\mathit{Succ}(v, x, y) \rightarrow \mathit{LT}(v, x, y) \tag{13}$$

$$\mathit{LT}(v, x, y) \wedge \mathit{LT}(v, y, z) \rightarrow \mathit{LT}(v, x, z) \tag{14}$$

With these relations, it is routine to define arithmetic relations such as $\mathit{Add}(o, a, b, c)$ (asserting that $c = a + b$ under the order o) and $\mathit{Bit}_i(o, a, b)$ (asserting that the b-th bit of the binary representation of a is i). We omit the details here.

Simulating Query Answering under O_Σ. With a partial successor relation and the related arithmetic relations, we are now in the position to define some DEDs to simulate the query answering of q under O_Σ and D.

Our encoding that implements the simulation of query answering is almost the same as that in Section 5.3 of (Zhang, Zhang, and You 2016). As proved by Zhang, Zhang, and You (see Proposition 6 of (2016)), all recursively enumerable OMQA-ontologies can be recognized by a certain class of Turing machines, called convergent 2-bounded nondeterministic Turing machines. Although the queries involved in that work are only boolean CQs, by a similar argument one can show that the result can be generalized to the case where boolean UCQs are involved. The only difference is that, to deal with UCQs, we have to change the format of input slightly. Due to the space limit, we omit the details here.

With the result mentioned above, we can then find a convergent 2-bounded nondeterministic Turing machine $\mathbb{M}_\Sigma$ to recognize O_Σ. By employing the DEDs defined in Section 5.3 of (Zhang, Zhang, and You 2016) (with a slight modification to specify the partial successor relation), we are then able to simulate the computation of $\mathbb{M}_\Sigma$ on the input (D, q).

To restore the result of the query answering, we use a binary relation symbol *Accept*. By $\mathit{Accept}(o, a)$ we mean that the machine $\mathbb{M}_\Sigma$ halts on input (D, q) with "accept", and o is the partial successor relation to implement the computation.

Generating Universal Models. By applying all the DEDs that we have constructed, we will obtain the class of boolean UCQs that are derivable from D under O_Σ. With such a class of UCQs, now our task is to construct a universal model set.

Given a class $\mathcal{D}$ of databases and a set C of constants, let

$$\bigoplus\nolimits_C \mathcal{D} = \bigcup \{D^* : D \in \mathcal{D}\}$$

where, for each database $D \in \mathcal{D}$, D^* is an isomorphic copy of D such that, for any pair of distinct databases $D_1, D_2 \in \mathcal{D}$, only constants from C will be shared by D_1^* and D_2^*.

Let D be a $\mathscr{D}$-database, and O an OMQA[UCQ]-ontology over $(\mathscr{D}, \mathscr{Q})$. Given a boolean UCQ q, let $\mathcal{D}_q$ denote the set consisting of $[p]$ for each disjunct (a boolean CQ) of q. Let

$$\Gamma(O, D) = \{\mathcal{D}_q : (D, q) \in O\}.$$

Let $\mathcal{U}(O, D)$ denote the set that consists of $\bigoplus_C H$ for each minimum hitting set H of $\Gamma(O, D)$, where $C = \mathit{adom}(D)$.

Proposition 8. *Let O be an OMQA[UCQ]-ontology over $(\mathscr{D}, \mathscr{Q})$, let D be a $\mathscr{D}$-database, and let q be a boolean $\mathscr{Q}$-UCQ such that $\mathit{const}(q) \subseteq \mathit{adom}(D)$. Then $(D, q) \in O$ iff $I \models q$ for all instances $I \in \mathcal{U}(O, D)$.*

Proof. Let Λ denote the set of all boolean UCQs p such that $(D, p) \in O$. We first show a property as follows.
Claim. $\Lambda \vDash q$ iff $I \models q$ for all instances $I \in \mathcal{U}(O, D)$.

Proof. First consider the direction of "only if". Suppose we have $\Lambda \vDash q$, and let I be any instance in $\mathcal{U}(O, D)$. We need to prove that $I \models q$. According to the definition of $\mathcal{U}(O, D)$, we know that there is a minimum hitting set H of $\Gamma(O, D)$ such that $I = \bigoplus_{\mathit{adom}(D)} H$. This implies that for each UCQ $q_0 \in \Lambda$ there is a disjunct p (which is a boolean CQ) of q_0 such that $[p]$ has an isomorphic copy in I. Consequently, we have that $I \models q_0$ for all boolean UCQs $q_0 \in \Lambda$. From the assumption that $\Lambda \vDash q$, we conclude $I \models q$ as desired.

Next let us turn to the direction of "if". Suppose $I \models q$ for all instances $I \in \mathcal{U}(O, D)$. Now our task is to prove that $\Lambda \vDash q$. Let J be an arbitrary instance such that $J \models p$ for all boolean UCQs $p \in \Lambda$. Take p as any boolean UCQ in Λ. W.l.o.g., we write p as the form $p_1 \vee \cdots \vee p_n$ where each p_i is a boolean CQ. Let $\varphi_p \in \{p_1, \ldots, p_n\}$ be any disjunct of p such that $J \models \varphi_p$. Such a disjunct always exists because $J \models p$. Suppose φ_p is of the form $\exists \vec{x}_p \psi_p$, where ψ_p is a conjunction of atoms and $\vec{x}_p$ a tuple of variables. Let s_p be an assignment such that $J \models \psi_p[s_p]$. Let H denote the set that consists of $[\varphi_p]$ for each UCQ $p \in \Lambda$, and let I denote the instance $\bigoplus_{\mathit{adom}(D)} H$. Let h be a mapping that maps the isomorphic copy of x in I to $s_p(x)$ if $p \in \Lambda$ and $x \in \vec{x}_p$, and maps each constant in $\mathit{adom}(D)$ to itself. Clearly, h is an $\mathit{adom}(D)$-homomorphism from I to J. By the assumption made in the begin of this paragraph, we know $I \models q$. As q is preserved under $\mathit{adom}(D)$-homomorphisms, we conclude $J \models q$, which completes the proof of the claim. □

According to the above claim, to prove the desired proposition, it suffices to show that $(D, q) \in O$ iff $\Lambda \vDash q$. The direction of "only if" is trivial since from $(D, q) \in O$ we already have $q \in \Lambda$. It thus remains to show the converse. Suppose $\Lambda \vDash q$. According to the compactness, there is a finite subset Λ_0 of Λ such that $\Lambda_0 \vDash q$. Let q_0 denote the conjunction of all UCQs in Λ_0. Obviously, q_0 is also a boolean UCQ. By the definition of Λ, we know that for each UCQ $q \in \Lambda_0$ we have $(D, q) \in O$. Since O is closed under query conjunctions, we obtain $(D, q_0) \in O$. By $\Lambda_0 \vDash q$, it also holds that $q_0 \vDash q$. Furthermore, by applying the closure property of O under query implications, we conclude $(D, q) \in O$, which completes the proof of the proposition immediately. □

With Proposition 8, we are now able to construct some DEDs to generate $\mathcal{U}(O, D)$. Some fresh relations are needed to access the encoding of a query. We use $\mathit{UCQ}(o, a)$ to denote that, under the order o, a is the representation (e.g., the Gödel number) of a boolean UCQ, and use $\mathit{Union}(o, a, b, c)$ to assert that, under the order o, the boolean UCQ a is the disjunction of a boolean CQ b and a boolean UCQ c. By $\mathit{CQ}(o, a)$ we mean that a is the encoding of some boolean CQ under the order o, and by $\mathit{QVar}(o, a, b)$ we means that

b is a variable that occurs in the boolean CQ encoded by a under the order o. Moreover, we assume $Q_1, \dots, Q_n$ enumerates all relation symbols in $\mathscr{Q}$. For $1 \leq i \leq k$, let $\mathit{HasQ}_i(o, a, \vec{t})$ denote that $Q_i(\vec{t})$ is an atom in the CQ encoded by a. It is easy to see that all these relations can be defined by standard arithmetic relations.

Now, we use the following DEDs to nondeterministically choose which disjunct of a boolean UCQ to be true:

$$\mathit{Accept}(v, x) \to \mathit{True}(v, x) \tag{15}$$

$$\mathit{True}(v, x) \wedge \mathit{Union}(v, x, y, z) \to \mathit{True}(v, y) \vee \mathit{True}(v, z) \tag{16}$$

where $\mathit{True}(o, a)$ states that the boolean CQ encoded by a under the order o is chosen to be true in the intended model.

To generate a copy of a boolean CQ in the intended universal model, we employ the following DEDs:

$$\mathit{CQ}(v, x) \wedge \mathit{QVar}(v, x, y) \to \exists z\, \mathit{Copy}(v, x, y, z) \tag{17}$$

$$\mathit{CQ}(v, x) \wedge \mathit{AD}(y) \to \mathit{Copy}(v, x, y, y) \tag{18}$$

$$\begin{split}\mathit{True}(v, x) \wedge \mathit{CQ}(v, x) \wedge \mathit{HasQ}_i(v, x, \vec{y})\\ \wedge\, \mathit{Copy}(v, x, \vec{y}, \vec{z}) \to Q_i(\vec{z})\end{split} \tag{19}$$

where $\mathit{Copy}(v, x, \vec{y}, \vec{z})$ denotes $\bigwedge_{1 \leq j \leq k} \mathit{Copy}(v, x, y_j, z_j)$ if $\vec{y} = y_1 \cdots y_k$, $\vec{z} = z_1 \cdots z_k$, and k is the arity of Q_i. Intuitively, the first rule generates a copy for each variable in the CQ q, the second one asserts that the constant in q will not change, and the third one then copy atoms in q into the universal model and implement some necessary substitutions.

Let $loc^\ell(\Sigma)$ denote the set of DEDs that we have defined in this section. From the encoding we then have

Lemma 9. $[\![loc^\ell(\Sigma)]\!]^{\mathscr{Q}}_{\mathscr{D},\mathscr{Q}} = O_\Sigma$.

Clearly, given the representation of any bound function ℓ and any finite set Σ of DEDs, constructing $loc^\ell(\Sigma)$ is computable. Let T_{F} consist of $loc^\ell(\Sigma)$ for each finite set Σ of DEDs and each bound function $\ell \in \mathsf{F}$. Since F is decidable, we know that T_{F} is also decidable. Let $\mathcal{L}_{\mathsf{F}}$ be the DED-language induced by T_{F}. By Lemmas 9, 5 and 6, $\mathcal{L}_{\mathsf{F}}$ must be universal for the family of F-local OMQA-languages.

Concluding Remarks

We have established the nonexistence of universal language for both FO-rewritable and PTIME-tractable OMQA. As a rescue, we also proposed a novel property, called the locality, as an approximation to the FO-rewritability, and proved that there is some language of DEDs which is universal for OMQA with bounded locality. In spite of the unnaturalness of the constructed language, we believe that the proposed property would shed light on finding natural universal languages, as well as on identifying new tractable languages.

Acknowledgements

Heng Zhang thanks the support of the National Natural Science Foundation of China (NO.61806102, NO.61972455). Zhiyong Feng thanks the support of the National Natural Science Foundation of China (NO.61672377). Guifei Jiang thanks the support of the National Natural Science Foundation of China (NO.61806102), National High Technology R&D Program of China (NO. 2018YFB0204304) and the Fundamental Research Funds for the Central Universities.

References

Baader, F.; Brandt, S.; and Lutz, C. 2005. Pushing the EL envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, 364–369.

Baget, J.; Leclère, M.; Mugnier, M.; and Salvat, E. 2011. On rules with existential variables: Walking the decidability line. *Artif. Intell.* 175(9-10):1620–1654.

Benedikt, M.; Leblay, J.; ten Cate, B.; and Tsamoura, E. 2016. *Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Berardi, D.; Calvanese, D.; and De Giacomo, G. 2005. Reasoning on UML class diagrams. *Artif. Intell.* 168(1-2):70–118.

Calì, A.; Gottlob, G.; and Lukasiewicz, T. 2012. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.* 14:57–83.

Calì, A.; Gottlob, G.; and Pieris, A. 2012. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.* 193:87–128.

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

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2013. Data complexity of query answering in description logics. *Artif. Intell.* 195:335–360.

Ebbinghaus, H., and Flum, J. 1995. *Finite model theory*. Perspectives in Mathematical Logic. Springer.

Ebbinghaus, H.; Flum, J.; and Thomas, W. 1994. *Mathematical logic (2. ed.)*. Undergraduate texts in mathematics. Springer.

Fagin, R.; Kolaitis, P. G.; Miller, R. J.; and Popa, L. 2005. Data exchange: Semantics and query answering. *Theoretical Computer Science* 336(1):89–124.

Gottlob, G.; Rudolph, S.; and Simkus, M. 2014. Expressiveness of guarded existential rule languages. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 27–38.

Immerman, N. 1999. *Descriptive complexity*. Graduate texts in computer science. Springer.

Lenzerini, M. 2002. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, 233–246.

Poggi, A.; Lembo, D.; Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Rosati, R. 2008. Linking data to ontologies. *J. Data Semantics* 10:133–173.

Rudolph, S., and Thomazo, M. 2015. Characterization of the expressivity of existential rule queries. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 3193–3199.

Vardi, M. Y. 1982. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC)*, 137–146.

Zhang, H.; Zhang, Y.; and You, J. 2015. Existential rule languages with finite chase: Complexity and expressiveness. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, 1678–1685.

Zhang, H.; Zhang, Y.; and You, J. 2016. Expressive completeness of existential rule languages for ontology-based query answering. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, 1330–1337.