Expressive Completeness of Existential Rule Languages for Ontology-based Query Answering

Heng Zhang[1] and **Yan Zhang**[2] and **Jia-Huai You**[3]
[1]School of Computer Science & Technology, Huazhong University of Technology & Science, Wuhan, China
hengzhang@hust.edu.cn
[2]School of Computing, Engineering & Mathematics, Western Sydney University, Penrith, Australia
[3]Department of Computing Science, University of Alberta, Edmonton, Canada

Abstract

Existential rules, also known as data dependencies in Databases, have been recently rediscovered as a promising family of languages for Ontology-based Query Answering. In this paper, we prove that disjunctive embedded dependencies exactly capture the class of recursively enumerable ontologies in Ontology-based Conjunctive Query Answering (OCQA). Our expressive completeness result does not rely on any built-in linear order on the database. To establish the expressive completeness, we introduce a novel semantic definition for OCQA ontologies. We also show that neither the class of disjunctive tuple-generating dependencies nor the class of embedded dependencies is expressively complete for recursively enumerable OCQA ontologies.

1 Introduction

Ontology-based Query Answering (OQA) has been a central issue in Ontological Reasoning and Knowledge Representation, and played an essential role in the Semantic Web, Data Modelling, and Data Exchange and Integration. Generally speaking, the problem of OQA may be stated as follows: Given a database D, an ontology (usually, a logical theory) Σ and a query $\boldsymbol{q}(\vec{x})$ with free variables $\vec{x}$, we need to find some answers $\vec{a}$ such that $D \cup \Sigma \models \boldsymbol{q}(\vec{a})$ holds. In this setting, D consists of a set of facts that have been observed, Σ represents the domain knowledge, and $\boldsymbol{q}$ describes the question about which we need to inquire. A major topic in OQA is to identify proper languages for expressing ontologies Σ.

Over the years, many languages have been developed for representing ontologies in OQA. In Description Logics, the DL-Lite family [Calvanese *et al.*, 2007], $\mathcal{EL}$ family [Baader *et al.*, 2005], and other variants have been proposed and extensively studied. More recently, existential rules, also known as data dependencies in Databases, and Datalog$^{\pm}$ in Knowledge Representation, have been rediscovered as a promising family of OQA languages, in which existential quantifiers, conjunctions and disjunctions may appear in the rule heads; see, e.g., [Baget *et al.*, 2011; Calì *et al.*, 2012; Alviano *et al.*, 2012; Bourhis *et al.*, 2013]. It is also worth noting some work on expressing ontologies by second-order logic, see, e.g., [Fagin *et al.*, 2005; Bourhis *et al.*, 2015].

While all these languages are of their specific features and hence are useful in different problem domains, some natural questions imposed to us are: Do we need to implement each of them? Can we have a universal language for OQA? On the other hand, as there are a large number of building blocks that have been introduced for specifying ontologies, it would be also interesting to know which of them is necessary for specifying ontologies in OQA. In this work, we try to answer these questions. In particular, we would like to know whether there is an expressively complete language for OQA.

Expressive completeness is one of the major principles for designing languages in Computer Science and Logic. For example, almost all the traditional programming languages, including Prolog, are known to be Turing complete; propositional logic can express all boolean functions; and by the well-known Lindström theorem, first-order logic is as expressive as any logic that enjoys both the compactness and the Löwenheim-Skolem property, see [Ebbinghaus *et al.*, 1994]. In Databases, the first-order language is shown to be complete for all the queries decidable in AC^0, and Datalog complete for the PTIME queries, see, e.g., [Immerman, 1999].

Very recently, by regarding existential rules as traditional database query languages, Gottlob *et al.* [2014b] showed that weakly-guarded existential rules capture the class of EXPTIME queries, and Rudolph and Thomazo [2015] proved that tuple-generating dependencies capture the class of recursively enumerable queries. Both of these interesting characteraztions shed new insights on understanding the expressiveness of existential rules. However, it is also worth noting that, towards a full understanding of the expressiveness of OQA languages, treating them as traditional query languages is not enough. For example, many languages in the DL-Lite family are UCQ-rewritable, and almost all known tractable OQA languages are Datalog-rewritable; however, it is clearly not convenient to use UCQ and Datalog as OQA languages.

In this paper, we investigate the expressiveness of OQA languages in a more precise sense, and focus on Ontology-based Conjunctive Query Answering (OCQA), where conjunctive queries are used as the underlying query language. Our main contributions are as follows: Firstly, we propose a semantic framework for OCQA (Section 3). Such framework is general enough to unify different OCQA languages and hence can be used as a basis for studying the expressiveness of these languages for our purpose. In this framework, we

then prove that disjunctive embedded dependencies (DEDs) precisely capture the class of recursively enumerable ontologies in OCQA (Section 4). Note that every set of DEDs can be rewritten as a set of disjunctive tuple-generating dependencies (DTGDs) and equality-generating dependencies (EGDs), two important classes of dependencies that have been extensively studied in OCQA. As an immediate consequence, our result implies a characterization of the expressiveness of DEDs as a traditional query language, which is similar to Rudolph and Thomazo's characterization [2015]. We also prove that neither DTGDs nor embedded dependencies (EDs) are expressively complete for recursively enumerable OCQA ontologies (Section 4). So the expressiveness between DEDs and DTGDs and EDs is explicitly differentiated. See [Zhang *et al.*, 2016] for a full version of this paper with more details.

2 Preliminaries

Databases, instances and structures. We assume Δ_c and Δ_v as infinite disjoint sets of *constants* and *variables*, respectively. *Terms* are constants or variables. A *relational schema* $\mathcal{R}$ is a set of *relation symbols*, and each of which is equipped with a natural number as its *arity*. Every *atom* over $\mathcal{R}$ is either an equality or a *relation atom* built upon terms and relation symbols in $\mathcal{R}$. A *fact* is a variable-free relation atom. Each *instance* over $\mathcal{R}$ is a set of facts over $\mathcal{R}$. Instances that are finite are called *databases*. For any instance I, let $adom(I)$ denote the set of constants occurring in I. Every *structure* $\boldsymbol{A}$ over $\mathcal{R}$ consists of a nonempty set $dom(\boldsymbol{A})$, named its *domain*, and an instance $I(\boldsymbol{A})$ over $\mathcal{R}$ such that $adom(I(\boldsymbol{A})) \subseteq dom(\boldsymbol{A})$. For convenience, sometimes an instance I will be regarded as a structure $\boldsymbol{A}$ such that $dom(\boldsymbol{A}) = adom(I)$.

Let $\mathcal{R}$ be a relational schema, $\boldsymbol{A}$ and $\boldsymbol{B}$ be structures over $\mathcal{R}$, and C be a subset of $dom(\boldsymbol{A}) \cap dom(\boldsymbol{B})$. A function $h : dom(\boldsymbol{A}) \to dom(\boldsymbol{B})$ is a *homomorphism* from $\boldsymbol{A}$ to $\boldsymbol{B}$ if, for all relation symbols $R \in \mathcal{R}$ and all constant tuples $\bar{a}$ of a proper length, $R(h(\vec{a})) \in I(\boldsymbol{B})$ if $R(\vec{a}) \in I(\boldsymbol{A})$. We write $h : \boldsymbol{A} \to_C \boldsymbol{B}$ if h is a homomorphism from $\boldsymbol{A}$ to $\boldsymbol{B}$ such that $h(c) = c$ for all $c \in C$, and write $h : \boldsymbol{A} \Rightarrow_C \boldsymbol{B}$ if $h : \boldsymbol{A} \to_C \boldsymbol{B}$ and h is injective. To simplify the presentation, h will be dropped from the notations if it is clear from the context, and C will be dropped if it is empty. Furthermore, if $h : \boldsymbol{A} \Rightarrow \boldsymbol{B}$ and an inverse g of h exists such that $g : \boldsymbol{B} \Rightarrow \boldsymbol{A}$, we call h an *isomorphism* from $\boldsymbol{A}$ to $\boldsymbol{B}$, and write $h : \boldsymbol{A} \cong \boldsymbol{B}$. Given a relational schema $\mathcal{S} \subseteq \mathcal{R}$ and an instance I over $\mathcal{R}$, let $I|_{\mathcal{S}}$ denote the set of facts involving relation symbols in $\mathcal{S}$ only.

Queries and existential rules. Let $\mathcal{R}$ be a relational schema. By a *query* over $\mathcal{R}$ we mean a first-order formula built upon atoms over $\mathcal{R}$ as usual. Given a query $\boldsymbol{q}$, let $const(\boldsymbol{q})$ denote the set of constants that occur in $\boldsymbol{q}$. A query is called *boolean* if it has no free variables. A *conjunctive query (CQ)* is a query of the form $\exists \vec{y}.\varphi(\vec{x}, \vec{y})$ where φ is a conjunction of relation atoms. Let BCQ be short for boolean CQ. For each BCQ $\boldsymbol{q}$, we can regard it as a database by renaming existential variables as special constants and removing all quantifiers; let $[\boldsymbol{q}]$ denote such a database. Given a BCQ $\boldsymbol{q}$, the *graph* of $\boldsymbol{q}$ is an undirected graph with each atom in $\boldsymbol{q}$ as a node, and with each pair of atoms as an edge if they share an existential variable. A BCQ is called *prime* if its graph is connected.

Every *disjunctive embedded dependency (DED)* over $\mathcal{R}$ is a first-order sentence σ of the following form:

$$\forall \vec{x} \forall \vec{y} (\phi(\vec{x}, \vec{y}) \to \exists \vec{z}_1.\psi_1(\vec{x}, \vec{z}_1) \vee \cdots \vee \exists \vec{z}_k.\psi_k(\vec{x}, \vec{z}_k)) \quad (1)$$

where $k \geq 1$, ϕ is a conjunction of relation atoms involving terms from $\vec{x} \cup \vec{y}$ only, each ψ_i is a conjunction of atoms 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 an *embedded dependency (ED)* if $k = 1$, called a *disjunctive tuple-generating dependency (DTGD)* if it is equality-free, and called a *tuple-generating dependency (TGD)* if it is both an ED and a DTGD. The right-hand side of the implication is called the *body*, and left-hand side the *head*. Given a set Σ of DEDs, a relation symbol in the relational schema of Σ is called *intensional* if it occurs in the head of some DED in Σ, and *extensional* otherwise. For simplicity, we will omit the universal quantifiers in σ, and replace "$\wedge$" in the body by ",".

Let D be a database over Σ and $\boldsymbol{q}$ be a database, a set of DEDs and a BCQ, respectively, with a common relational schema $\mathcal{R}$. We write $D \cup \Sigma \models \boldsymbol{q}$ if, for all instances I over $\mathcal{R}$ such that $D \subseteq I$, if $I \models \Sigma$ then $I \models \boldsymbol{q}$, where the satisfaction relation $\models$ is defined as usual. Let $\boldsymbol{p}(\vec{x})$ be a CQ over $\mathcal{R}$ with an n-tuple of free variables, the set of *answers of* $\boldsymbol{p}$ *w.r.t.* D *under* Σ, denoted $Ans(D, \Sigma, \boldsymbol{p})$, is defined as the set of all n-tuples $\vec{a}$ of constants in $adom(D)$ such that $D \cup \Sigma \models \boldsymbol{p}(\vec{a})$.

A modular lemma. Below let us give a simple property that will be useful to verify the correctness of the OQA programming. We need to recall some notions first. An instance I is called a *minimal model* of a set Σ of DEDs if $I \models \Sigma$ and there exists no instance $J \subset I$ such that $J \models \Sigma$. Moreover, Σ is *well-founded* if for every instance J that satisfies Σ, there is a minimal model I of Σ such that $I \subseteq J$.

Lemma 1 (Modular lemma)**.** *Let $\boldsymbol{q}$ be a BCQ, and let Σ and Γ be two sets of DEDs such that Σ is well-founded and possibly involves constants, no relation symbol from the relational schema of Σ is intensional in Γ, and all relation symbols from the relational schema of $\boldsymbol{q}$ are intensional in Γ. Then $\Sigma \cup \Gamma \models \boldsymbol{q}$ iff $I \cup \Gamma \models \boldsymbol{q}$ for all minimal models I of Σ.*

3 OCQA ontologies

As mentioned earlier, our task is to establish the expressive completeness of languages for OQA. To do this, we first have to answer the following questions firstly: What is an ontology in OQA? Can we have a general and semantic definition for OQA ontologies so that different languages can be unified in the same framework? Below we will address these issues.

3.1 Some general properties

Most languages that have been proposed for OQA are logical languages, including description logics, first-order logic, and sometimes fragments of higher-order logic. So, to study general properties that should be enjoyed by OQA ontologies, it is natural to consider those for ontologies as logical theories.

Instead of considering a specific logical language, we will regard every logical theory as a class of structures.

Definition 1. Let $\mathcal{R}$ be a relational schema. An *abstract theory* over $\mathcal{R}$ is a class Φ of structures over $\mathcal{R}$ that is closed under isomorphisms, i.e., if $\boldsymbol{A} \cong \boldsymbol{B}$ and $\boldsymbol{A} \in \Phi$, then $\boldsymbol{B} \in \Phi$.

It is clear that, given a concrete theory Σ in any of the logical languages proposed to represent ontologies, the class of all models of Σ is an abstract theory even if Σ is infinite.

Let $\mathcal{R}$ be a relational schema. Let D, Φ and $\boldsymbol{q}(\vec{x})$ be a database, an abstract theory and a query over $\mathcal{R}$, respectively. Suppose k is the length of $\vec{x}$. We let $Ans(D, \Phi, \boldsymbol{q})$ denote

$$\{\vec{a} \in adom(D)^k : \forall \boldsymbol{A} \in \Phi\,(\boldsymbol{A} \models D \Longrightarrow \boldsymbol{A} \models \boldsymbol{q}(\vec{a}))\}. \quad (2)$$

Slightly abusing the notation, we will not distinguish between abstract and concrete theories (e.g., TGD sets) if no confusion occurs. For instance, notations like $D \cup \Phi \models \boldsymbol{q}$ will be used.

The following proposition gives us some useful properties.

Proposition 2. *Let $\mathcal{R}$ be a relational schema; Φ be an abstract theory over $\mathcal{R}$; D, D' be databases over $\mathcal{R}$; and $\boldsymbol{q}, \boldsymbol{q}'$ are boolean queries over $\mathcal{R}$. Then all the following hold:*

1. *If $D \cup \Phi \models \boldsymbol{q}$ and $D \cup \Phi \models \boldsymbol{q}'$ then $D \cup \Phi \models \boldsymbol{q} \wedge \boldsymbol{q}'$.*
2. *If $\boldsymbol{q} \models \boldsymbol{q}'$ and $D \cup \Phi \models \boldsymbol{q}$ then $D \cup \Phi \models \boldsymbol{q}'$.*
3. *If $D \Rightarrow_C D'$ and $D \cup \Phi \models \boldsymbol{q}$ then $D' \cup \Phi \models \boldsymbol{q}$, where C denotes the set of constants occurring in $\boldsymbol{q}$.*

3.2 A semantic definition of OCQA ontology

With the properties we have presented, we are now in the position to give the semantic definition for ontologies in OQA. Note that our semantic definition is a refinement of the notion proposed by Arenas *et al.* [2014] where ontologies are regarded as arbitrary sets of database-query-answer triples.

Definition 2. Let $\mathcal{D}, \mathcal{Q}$ be relational schemas. Every *quasi-OCQA ontology* over $(\mathcal{D}, \mathcal{Q})$ is a set of pairs $(D, \boldsymbol{q})$ where D is a database over $\mathcal{D}$ and $\boldsymbol{q}$ is a BCQ over $\mathcal{Q}$ with $const(\boldsymbol{q}) \subseteq adom(D)$. A quasi-OCQA ontology O over $(\mathcal{D}, \mathcal{Q})$ is called an *OCQA ontology* if all the following conditions hold:

1. O is *closed under query conjunctions*; that is, if both $(D, \boldsymbol{q}) \in O$ and $(D, \boldsymbol{q}') \in O$ then $(D, \boldsymbol{q} \wedge \boldsymbol{q}') \in O$.
2. O is *closed under query implications*; that is, if $\boldsymbol{q} \models \boldsymbol{q}'$ and $(D, \boldsymbol{q}) \in O$ then $(D, \boldsymbol{q}') \in O$.
3. O is *closed under injective database homomorphisms*; that is, if $D \Rightarrow_C D'$ and $(D, \boldsymbol{q}) \in O$ then $(D', \boldsymbol{q}) \in O$, where C denotes the set of constants occurring in $\boldsymbol{q}$.

Remark 1. Quasi-OCQA ontologies defined above provide a rather relaxed way to define abstract OCQA ontologies. This notion is slightly more restricted than Arenas *et al.*'s notion (i.e., database-query-answer pairs, see [Arenas *et al.*, 2014]) as we require: (i) all the relational symbols involved in the database (resp., query) must belong to a given relational schema $\mathcal{D}$ (resp., $\mathcal{Q}$), and (ii) each constant involved in the query must appear in the corresponding database. Since $\mathcal{D}$ and $\mathcal{Q}$ could be the same, the first assumption actually does not lose any generality. In addition, as usual in Databases, it is also reasonable to assume that only constants that occur in the database will be involved in the query answering.

Remark 2. It is worth noting that the quasi-OCQA ontology is a very loose notion to define abstract OCQA ontologies. In fact, it is not hard to show that there are many quasi-OCQA ontologies that are not definable by any logical theory. This is the reason why we define the notion of OCQA ontologies.

Definition 3. Let $\mathcal{D}, \mathcal{Q}, \mathcal{R}$ be relational schemas such that $\mathcal{R} \supseteq \mathcal{D} \cup \mathcal{Q}$, and Φ be an abstract theory over $\mathcal{R}$. We define $Sem(\Phi, \mathcal{D}, \mathcal{Q})$ as the set of ordered pairs $(D, \boldsymbol{q})$ such that D is a database over $\mathcal{D}$, $\boldsymbol{q}$ is a BCQ over $\mathcal{Q}$, and $D \cup \Phi \models \boldsymbol{q}$. Given an OCQA ontology O over $(\mathcal{D}, \mathcal{Q})$, we say that O is *defined by Φ over* $(\mathcal{D}, \mathcal{Q})$ if $O = Sem(\Phi, \mathcal{D}, \mathcal{Q})$.

With the above definition, any logical theory Φ can be related to abstract OCQA ontologies. By Proposition 2, we know that $Sem(\Phi, \mathcal{D}, \mathcal{Q})$ is always an OCQA ontology.

Example 1. Let Σ be a set consisting of only the DED:

$$\bigwedge_{0 \le i < 4} (\mathsf{A}(x_i, x_{s(i,4)}) \wedge \mathsf{A}(x_{s(i,4)}, x_i)) \rightarrow \mathsf{Goal} \vee \bigvee_{0 \le i < j < 4} x_i = x_j \quad (3)$$

where $s(i, k)$ denotes $i + 1$ if $i < k$, and 0 otherwise. Let D_k be the database $\{\mathsf{A}(a_i, a_{s(i,k)}) : 0 \le i < k\} \cup \{\mathsf{A}(a_{s(i,k)}, a_i) : 0 \le i < k\}$. It is a routine task to check that $D_4 \cup \Sigma \models \mathsf{Goal}$, and also easy to see that $D_3 \cup \Sigma \models \mathsf{Goal}$ does not hold.

Let $\mathcal{D} = \{\mathsf{A}\}$ and $\mathcal{Q} = \{\mathsf{Goal}\}$. Then $Sem(\Sigma, \mathcal{D}, \mathcal{Q})$ is an OCQA ontology which consists of pairs (D, Goal) for all databases D over $\mathcal{D}$ with a subset that is isomorphic to D_4.□

In Definition 3, only BCQs are considered. However, in query answering, non-boolean CQs will be used. The following fact shows that no generality will be lost in our setting.

Proposition 3. *Let $\mathcal{D}, \mathcal{Q}, \mathcal{R}$ be relational schemas such that $\mathcal{D} \cup \mathcal{Q} \subseteq \mathcal{R}$, and Φ, Ψ be abstract theories over $\mathcal{R}$ such that $Sem(\Phi, \mathcal{D}, \mathcal{Q}) = Sem(\Psi, \mathcal{D}, \mathcal{Q})$. Then for all databases D over $\mathcal{D}$ and CQs $\boldsymbol{q}$ over $\mathcal{Q}$, $Ans(D, \Phi, \boldsymbol{q}) = Ans(D, \Psi, \boldsymbol{q})$.*

3.3 Expressive (in)completeness

Next, let us consider how to express arbitrary OCQA ontologies by DEDs. Due to a theoretical interest, we consider infinite sets of DEDs here. Such an investigation will provide a rational justification for our semantic definition.

Proposition 4. *Let $\mathcal{D}, \mathcal{Q}$ be two disjoint relational schemas, and O be a quasi-OCQA ontology over $(\mathcal{D}, \mathcal{Q})$. Then O can be defined by a possibly infinite set of DEDs iff it is an OCQA ontology.*

Proof (Sketch). The direction "only-if" is by Proposition 2. To show the converse, for each database-query pair from O, it is easy to construct a DED to generate its answers. □

In the above result, the DED set could be infinite. A natural question arises as whether all OCQA ontologies can be defined by a finite set of DEDs. For quasi-OCQA ontologies, the answer is negative. This is because there is some quasi-OCQA ontology which is highly undecidable, but query answering for finite sets of DEDs is recursively enumerable.[1] Naively, one might hope that the properties defining OCQA ontologies will get rid of the high undecidability. However, the following result tells us that this case will never happen.

Proposition 5. *There is some (highly undecidable) OCQA ontology that is not definable by any first-order sentence.*

Proof (Sketch). One can encode the recurring domino problem (Σ^1_1-hard [Harel, 1986]) by some OCQA ontology. Note that query answering with first-order sentences is in Σ^0_1. □

[1] Note that DEDs are first-order sentences and that the inference validity of first-order logic is recursively-enumerable complete.

4 Recursively enumerable OCQA ontologies

In the last section, we have proved that the language consisting of finite sets of TGDs is not enough to capture the entire class of OCQA ontologies. This is not surprising because in the semantic definition of OCQA ontologies we do not consider whether the OCQA ontologies are realizable or not. In this section, we will focus on realizable OCQA ontologies, or more explicitly, recursively enumerable OCQA ontologies.

4.1 Machines that recognize OCQA ontologies

Before giving the characterizations, we first recall some notions and present a useful property. In the rest of this section, every *nondeterministic Turing machine (NTM)* is equipped with a read-only input tape and a read-write work tape. The head on the input tape will move right in all stages. Formally, an NTM M is defined by a 6-tuple $(S, s_0, A, \Gamma, \Box, \delta)$, where

- S is a finite and nonempty set of states, $s_0 \in S$ is the initial state, and $A \subseteq S$ is a set of accepting states;
- Γ denotes the alphabet $\{0, 1, \Box, \sharp, \bar{0}, \bar{1}\}$,[2] where "$\Box$" will be used as the blank symbol;
- $\delta : (S \times \Gamma \times \Gamma) \to \mathcal{P}(S \times \Gamma \times \{L, R\})$ is a transition function, where $\mathcal{P}(X)$ denotes the power set of X if X is a set. Every *legal transition* of M is a possible transition according to the transition function δ defined as usual.

Given an integer $k > 0$, we say that M is *k-bounded* if for every triple $(s, a, b) \in S \times \Gamma \times \Gamma$ we have that $|\delta(s, a, b)| \leq k$. In particular, if M is 1-bounded, we call M *deterministic*, or directly say that M is a *deterministic Turing machine (DTM)*. It is well-known that every NTM can be simulated by a DTM.

Given a natural number k, let $b(k)$ denote the binary representation (using "0" and "1") of k. We also fix $\mathcal{D}$ and $\mathcal{Q}$ as two relational schemas, D as a database over $\mathcal{D}$, and $\boldsymbol{q}$ as a BCQ over $\mathcal{Q}$. For $* \in \{d, q\}$, we let w_* and n_* denote the maximum arity and the number of relation symbols in $\mathcal{R}_*$, respectively. Let c denote the size of $adom(D)$, and e denote the number of existential quantified variables in $\boldsymbol{q}$. Now we encode $(D, \boldsymbol{q})$ by the string, denoted by $[\![D, \boldsymbol{q}]\!]$, as follows:

$$b(w_d)\sharp b(n_d)\sharp b(c)\sharp\overline{tvt(D)}\sharp b(w_q)\sharp b(n_q)\sharp b(e)\sharp tvt(\boldsymbol{q}) \quad (4)$$

where $tvt(D)$ is a binary string encoding all truth values of ground atoms (built upon $adom(D)$ and $\mathcal{D}$) w.r.t. D; $tvt(\boldsymbol{q})$ is defined similarly by regarding $\boldsymbol{q}$ as a special database; and for a given string str, let $\overline{str}$ denote the string obtained from str by substituting $\bar{0}$ and $\bar{1}$ for 0 and 1, respectively.

Example 2. Suppose $\mathcal{D} = \{\mathsf{D}, \mathsf{E}\}$ and $\mathcal{Q} = \{\mathsf{Q}\}$, where D, E are unary, and Q is binary. Suppose D is a database consisting of a single fact $\mathsf{E}(a)$, and $\boldsymbol{q}$ denotes the BCQ $\exists x \mathsf{Q}(x, a)$. Then we can encode $(D, \boldsymbol{q})$ by the string

$$1\sharp 10\sharp 1\sharp\bar{0}\bar{1}\sharp 10\sharp 1\sharp 1\sharp 0010 \quad (5)$$

where by the string "$\bar{0}\bar{1}$" we mean that the fact $\mathsf{D}(a)$ is false in D, and the fact $\mathsf{E}(a)$ is true in D; by the string "0010" (in the last block) we mean that the atom $\mathsf{Q}(x, a)$ has an occurrence in $\boldsymbol{q}$, while atoms $\mathsf{Q}(a, a), \mathsf{Q}(a, x)$ and $\mathsf{Q}(x, x)$ have not. $\Box$

[2] For a technical reason, we use two additional symbols "$\bar{0}$" and "$\bar{1}$" to represent special binary strings. It is easy to see that every NTM defined here can be simulated by one with a standard setting.

Definition 4. Let $\mathcal{D}$ and $\mathcal{Q}$ be relational schemas, and O be an OCQA ontology over $(\mathcal{D}, \mathcal{Q})$. Then O is *recognized* by an NTM M if, for all databases D over $\mathcal{D}$ and all BCQs $\boldsymbol{q}$ over $\mathcal{Q}$, $(D, \boldsymbol{q}) \in O$ iff M accepts $[\![D, \boldsymbol{q}]\!]$; O is *recursively enumerable* if O is recognized by some deterministic NTM.

To establish the desired expressive completeness, we need to simulate Turing machines by existential rules. The first obstacle to implement such a simulation is the lack of negation in the bodies of existential rules so that it would be difficult to represent the information "a fact is false in the database". Below we present a novel technique to address this issue.

Let M be an abitrary NTM $(S, s_0, A, \Gamma, \Box, \delta)$. We call M *convergent* if $\delta(s, \bar{0}, b) \subseteq \delta(s, \bar{1}, b)$ for all states $s \in S$ and all symbols $b \in \Gamma$. In other words, if M is convergent, then every legal transition of M on the input symbol $\bar{0}$ should be also a legal transition of M on the input symbol $\bar{1}$.

Now let us explain the general idea of representing the negative information in a given database. Suppose D is a relation symbol in the database schema and τ is a transition of some convergent NTM M. Let $\phi_\tau(\vec{x})$ be a formula that describes the preconditions to implement the transition τ (except for the information about the truth of $\mathsf{D}(\vec{x})$), and $\psi_\tau(\vec{x})$ be a formula that defines the effect of implementing τ. If $\mathsf{D}(\vec{x})$ is true in the database, one can simulate the transition τ by the rule

$$\mathsf{D}(\vec{x}), \phi_\tau(\vec{x}) \to \psi_\tau(\vec{x}). \quad (6)$$

For the case where $\mathsf{D}(\vec{x})$ is false in the database, by the encoding of the input we know that the bit which encodes the truth of $\mathsf{D}(\vec{x})$ is $\bar{0}$. Since M is convergent, τ is legal for the case where $\mathsf{D}(\vec{x})$ is false implies that τ is legal for the case where $\mathsf{D}(\vec{x})$ is true. Thus we can now simulate τ by the rule

$$\phi_\tau(\vec{x}) \to \psi_\tau(\vec{x}). \quad (7)$$

According to the definition of the convergent NTM, it is easy to see that Rules (6) and (7) will always implement consistent actions. Note that no negation is involved in the simulation.

Now, let us present an interesting observation that will play an important role in establishing the main characterization.

Proposition 6. *Every recursively enumerable OCQA ontology can be recognized by a convergent, 2-bounded NTM.*

Proof. Let O be a recursively enumerable OCQA ontology. Then there is a DTM, say M, that recognizes O. We assume $M = (S, s_0, A, \Gamma, \Box, \delta)$. Let δ_c be defined as follows:

$$\delta_c(s, a, b) = \begin{cases} \delta(s, \bar{1}, b) \cup \delta(s, \bar{0}, b) & \text{if } a = \bar{1}, \\ \delta(s, a, b) & \text{otherwise.} \end{cases} \quad (8)$$

Now let $M_c = (S, s_0, A, \Gamma, \Box, \delta_c)$. It is easy to see that M_c is a convergent and 2-bounded NTM. So, it remains to show that M_c also recognizes O. Given any input, it is clear that all accepting paths of M are accepting paths of M_c. Thus, M_c accepts all inputs $[\![D, \boldsymbol{q}]\!]$ such that $(D, \boldsymbol{q}) \in O$.

Conversely, let $(\mathcal{D}, \mathcal{Q})$ be the relational schema pair of O, D be a database over $\mathcal{D}$, and $\boldsymbol{q}$ be a BCQ over $\mathcal{Q}$ such that M_c accepts the input $ins = [\![D, \boldsymbol{q}]\!]$. We want to show that $(D, \boldsymbol{q}) \in O$. Let P be an accepting path that accepts ins. Let ins_0 be a string defined as follows: If $\bar{1}$ is scanned in the

i-th stage of P, and the next transition is not implemented by original actions in δ, then let the i-th symbol of ins_0 be $\bar{0}$, otherwise let it be the same as that of ins. It is easy to see that ins_0 encodes a pair $(D', \boldsymbol{q})$ such that $D' \Rightarrow_C D$ where C denotes $adom(D)$. Moreover, it is not hard to verify that ins_0 will be accepted by the original machine M, which means that $(D', \boldsymbol{q}) \in O$. Since O is closed under injective database homomorphisms, we obtain that $(D, \boldsymbol{q}) \in O$ as desired. □

4.2 Expressive (in)completeness

In the following we present the main theorem, which states that the disjunctive embedded dependencies are expressively complete for recursively enumerable OCQA ontologies.

Theorem 7. *Let $\mathcal{D}, \mathcal{Q}$ be two disjoint relational schemas, and O be a quasi-OCQA ontology over $(\mathcal{D}, \mathcal{Q})$. Then O can be defined by a finite set of DEDs iff it is a recursively enumerable OCQA ontology.*

We leave the proof of this theorem to the next section.

Now, let us first present an interesting implication of our characterization. Some notions are needed. Fix $\mathcal{R}$ as a relational schema. A *database query* over $\mathcal{R}$ is a class of databases over $\mathcal{R}$ that is closed under isomorphisms. Similar to that in [Rudolph and Thomazo, 2015], let Goal be a special nullary relation symbol which does not belong to $\mathcal{R}$. A database query Q is called *defined by* a set Σ of DEDs if we have $D \in Q$ iff $D \cup \Sigma \models \mathsf{Goal}$. By Theorem 7, we then have a characterization for classical database queries, which is similar to Rudolph and Thomazo's characterization [2015]:

Corollary 8. *A database query Q is defined by a finite DED set iff Q is recursively enumerable and closed under injective homomorphisms, i.e., $D \in Q$ and $D \to D'$ implies $D' \in Q$.*

Next, let us consider the following question: Can the main theorem be established for a simpler language? For example, can equalities (resp., disjunctions) be removed so that DTGDs (resp., EDs) are still expressively complete? The following propositions show that, to assure the expressive completeness, neither disjunctions nor equalities can be removed.

Let us first define a notion. An OCQA ontology O over a relational schema pair $(\mathcal{D}, \mathcal{Q})$ is called *first-order rewritable* if for all BCQ $\boldsymbol{q}$ over $\mathcal{Q}$ there is a first-order sentence $\boldsymbol{q}_O$ such that $(D, \boldsymbol{q}) \in O$ iff $D \models \boldsymbol{q}_O$ for all databases D over $\mathcal{Q}$. With this notion, we are then in the position to present the results.

Proposition 9. *There is a first-order rewritable OCQA ontology that cannot be defined by any finite set of DTGDs.*

Proposition 10. *There is a first-order rewritable OCQA ontology that cannot be defined by any finite set of EDs.*

Proposition 9 can be proved by showing that every OCQA ontology defined by DTGDs is closed under database homomorphisms. Example 1 gives a first-order rewritable OCQA ontology that is not closed under database homomorphisms. The next proposition can be proved by showing that the complement of every OCQA ontology defined by EDs is closed under direct products; however first-order rewritable OCQA ontologies that do not have this property can be easily found.

Remark 3. The above incompleteness results show why disjunctions and equalities should be considered in identifying new first-order rewritable and tractable OCQA languages.

5 Proof of the main theorem

From now on, let us focus on the proof of Theorem 7. The direction "only-if" follows from Proposition 2 and the well-known fact that the inference validity of first-order logic is recursively-enumerable complete. So, in this section we only consider the converse, which will be proved by constructing a finite set of DEDs. We first explain the general idea of the construction, and then implement it step by step.

5.1 General idea of the proof

In Databases, to show a language is expressively complete for a class of database queries in a certain level of complexity, a usual way is by constructing logical theories to simulate Turing machines, see, e.g., [Vardi, 1982; Rudolph and Thomazo, 2015]. However, it should be noted that establishing the expressive completeness of OQA languages is significantly different from that of classical querying languages.

The main difficulty is as follows: Although the BCQ is one part of the input for the query answering, the logical theory that simulates the query answering actually cannot access the BCQ, which means that, instead of focusing on one decision problem, we have to treat an infinite number of decision problems. Note that there are an infinite number of BCQs.

Fortunately, we have found a novel technique to overcome these obstacles. To explain it, we first define some notations, and then present a property. Let $\mathcal{R}$ be a relational schema, $\mathscr{D}$ be a class of databases over $\mathcal{R}$, and C be a set of constants. We let $\bigoplus_C \mathscr{D}$ denote the instance $\bigcup\{D^* : D \in \mathscr{D}\}$ where, for every $D \in \mathscr{D}$, D^* is an isomorphic copy (a database over $\mathcal{R}$) of D such that, for any pair of distinct databases D_1, D_2 in $\mathscr{D}$, only constants from C will be shared by D_1^* and D_2^*.

The following is a key lemma to our proof.

Proposition 11. *Let $\mathcal{D}, \mathcal{Q}$ be relational schemas, and O be an OCQA ontology over $(\mathcal{D}, \mathcal{Q})$. Then for all databases D over $\mathcal{D}$ and BCQs $\boldsymbol{q}$ over $\mathcal{Q}$, $(D, \boldsymbol{q}) \in O$ iff $U_O(D) \models \boldsymbol{q}$, where $U_O(D) = \bigoplus_C \{[\boldsymbol{q}'] : (D, \boldsymbol{q}') \in O\}$ and $C = adom(D)$.*

Proof. The direction "only-if" is trivial; so we only show the converse. Let $\boldsymbol{q}$ be a BCQ over $\mathcal{Q}$ and D be a database over $\mathcal{D}$ such that $U_O(D) \models \boldsymbol{q}$. By definition, there exists a homomorphism h from $[\boldsymbol{q}]$ to $U_O(D)$ such that $h(c) = c$ for all constants c from C. Let $\boldsymbol{q}'$ be a BCQ obtained from $\boldsymbol{q}$ by substituting $h(x)$ for all variables x such that $h(x) \in C$, and let h' be a restriction of h to all variables and constants occurring in $\boldsymbol{q}'$. Then it is clear that $\boldsymbol{q}' \models \boldsymbol{q}$ and that h' is a homomorphism from $[\boldsymbol{q}']$ to $U_O(D)$. Without loss of generality, let us write $\boldsymbol{q}'$ in the form $\boldsymbol{q}'_1 \wedge \cdots \wedge \boldsymbol{q}'_k$, where $k \geq 1$ and each $\boldsymbol{q}'_i$ is prime (see Section 2 for the definition). Take $i \in \{1, \ldots, k\}$. Obviously, h' is a homomorphism from $[\boldsymbol{q}'_i]$ to $U_O(D)$. Since $\boldsymbol{q}'_i$ is prime, by the definition of $U_O(D)$ we know that there is at least one BCQ $\boldsymbol{p}_i$ such that $(D, \boldsymbol{p}_i) \in O$ and that $h'(\boldsymbol{q}'_i)$ is contained in the disjoint copy of $\boldsymbol{p}_i$ in $U_O(D)$. From the latter we conclude that $\boldsymbol{p}_i \models \boldsymbol{q}'_i$. Since O is closed under query implications, we infer that $(D, \boldsymbol{q}'_i) \in O$. Furthermore, since O is closed under query conjunctions, we obtain that $(D, \boldsymbol{q}') \in O$. As we have proved previously, it holds that $\boldsymbol{q}' \models \boldsymbol{q}$. Again by the query implication closure of O, we conclude that $(D, \boldsymbol{q}) \in O$, which completes the proof. □

With the property above, we call $U_O(D)$ a *universal model* of O w.r.t. D. Now, let us explain the general idea of our proof. From now on, let us fix $\mathcal{D}$ and $\mathcal{Q}$ as two disjoint relational schemas, and O as a recursively enumerable OCQA ontology over $(\mathcal{D}, \mathcal{Q})$. Our task is to construct a finite set Σ of DEDs such that, for any given database D over $\mathcal{D}$, $U_O(D)$ is isomorphic to the minimal model of $D \cup \Sigma$.

Now, we show how to construct the set Σ of DEDs. Suppose M is a Turing machine which recognizes O. For all possible BCQs $\boldsymbol{q}$ over $\mathcal{Q}$, we let Σ work as follows:

1. Simulate the computation of M on input $ins = [\![D, \boldsymbol{q}]\!]$.
2. If M accepts ins then copy $[\boldsymbol{q}]$ to the intended model.

To implement the simulation of M, we have to define natural numbers on the intended domain, which can be done in DEDs easily. Another difficulty is to encode negative information in the database as existential rules have no negations (see [Rudolph and Thomazo, 2015]). Thanks for the power of Proposition 6, this again can be overcome in a natural way.

5.2 Defining numbers and arithmetic relations

To define numbers and arithmetic relations, we have to generate a successor relation first. In particular, we require that the successor relation will travel over all the constants from the database so that every fact in the database could have a fixed encoding. Let DC be a fresh unary relation symbol that collects all the constants occurring in the intended database, which can be defined by the following DEDs:

$$\mathsf{D}_i(x_1, \ldots, x_k) \to \mathsf{DC}(x_1) \wedge \cdots \wedge \mathsf{DC}(x_k) \tag{9}$$

for all relation symbols D_i from the schema $\mathcal{D}$, where k denotes the arity of D_i. Let Σ_s denote the set that consists of all the DEDs which define DC and all the following DEDs:

$$\mathsf{DC}(x), \mathsf{DC}(y) \to \mathsf{LT}(x, y) \vee x = y \vee \mathsf{LT}(y, x) \tag{10}$$
$$\mathsf{LT}(x, y), \mathsf{LT}(y, z) \to \mathsf{LT}(x, z) \tag{11}$$
$$\mathsf{LT}(x, x) \to \mathsf{Undesired} \tag{12}$$
$$\mathsf{DC}(x), \mathsf{DC}(y), \mathsf{LT}(x, y) \to \mathsf{NotMin}(y) \tag{13}$$
$$\mathsf{DC}(x) \to \mathsf{Min}(x) \vee \mathsf{NotMin}(x) \tag{14}$$
$$\mathsf{DC}(x), \mathsf{DC}(y), \mathsf{LT}(x, y), \mathsf{LT}(y, z) \to \mathsf{LTNotSucc}(x, z) \tag{15}$$
$$\mathsf{DC}(x), \mathsf{DC}(y), \mathsf{LT}(x, y) \to \mathsf{Succ}(x, y) \vee \mathsf{LTNotSucc}(x, y) \tag{16}$$
$$\mathsf{Succ}(x, y) \to \exists z.\mathsf{Succ}(y, z) \wedge \mathsf{LT}(y, z) \tag{17}$$
$$\mathsf{Succ}(x, y) \to \mathsf{Num}(x) \wedge \mathsf{Num}(y) \tag{18}$$

where $\mathsf{Succ}(x, y)$ asserts that y is an immediate successor of x, and $\mathsf{LT}(x, y)$ states that x is strictly less than y. In this way, a linear order would be arbitrarily generated by disjunctions in DED (10) and the existential quantifier in DED (17). Note that the linear order generated by such disjunctions and existential quantifier is not always good – sometimes a cycle could be there. In this case, we simply set the nullary relation $\mathsf{Undesired}$ true. As we will see later, this flag will lead to giving up the intended instance under the current focus.

If the flag $\mathsf{Undesired}$ is false, by the transitivity and irreflexivity, we know that LT is a linear order. Thus, we have:

Proposition 12. *Let D be a database over $\mathcal{D}$. Then $D \cup \Sigma_s$ is well-founded, and for each minimal model I of $D \cup \Sigma_s$ with* $\mathsf{Undesired} \notin I$, Succ *defines an infinite successor relation on $adom(I)$,* Min *defines the minimum element w.r.t.* Succ*, and* Num *defines the set of all elements from $adom(I)$.*

With the above relations, we can define relations such as Add, Mul, Bit_b and Len, where $b \in \{0, 1\}$, $\mathsf{Add}(x, y, z)$ ($\mathsf{Mul}(x, y, z)$, respectively) asserts that z is the sum (product, respectively) of x and y, $\mathsf{Bit}_b(x, y)$ asserts that the y-th bit of the binary representation of x is b. All of these can be defined in a routine way. For example, the addition relation can be defined by the following DEDs:

$$\mathsf{Num}(x), \mathsf{Min}(y) \to \mathsf{Add}(x, y, x) \tag{19}$$
$$\mathsf{Add}(x, y, z), \mathsf{Succ}(y, u), \mathsf{Succ}(z, v) \to \mathsf{Add}(x, u, v) \tag{20}$$

Finally, we let Σ_{num} denote the union of Σ_s and the set of all DEDs that define the mentioned arithmetic relations.

5.3 Simulating Turing machine

Now, we are ready to define some DEDs to simulate the Turing machine. According to Proposition 6, we only need to simulate a convergent, 2-bounded NTM M. As mentioned previously, we will simulate the computations of M on all possible input queries in parallel. Towards this end, we introduce a number of relation symbols ITape_a, WTape_a, IHead, WHead and State_s, where a is a symbol in the alphabet, and s is a state. For example, $\mathsf{ITape}_a(x, y)$ asserts that, in the computation for the BCQ encoded by y, the symbol written on the input tape at position x is a; $\mathsf{State}_s(x, y, z)$ asserts that, in the computation path (encoded by) y for BCQ (encoded by) z, the state of M in time x is s. Note that, as M is 2-bounded, the (nondeterministic) choices of all stages can be represented by a binary string in an obvious way.

The simulation of M consists of three parts as follows.

Initialization. This part includes specifying the initial state, copying the string encoding a database-query pair to the input tape, and filling unused cells with the blank symbol. We only focus on the second one here. The other two are trivial.

Suppose relation symbols allowed in database (resp., BCQ) list as $\mathsf{D}_0, \ldots, \mathsf{D}_{m-1}$ (resp., $\mathsf{Q}_0, \ldots, \mathsf{Q}_{n-1}$). We first introduce some fresh relation symbols PosD_i and PosQ_j, where $0 \leq i < m$, $0 \leq j < n$, PosD_i (resp., PosQ_j) is of arity $k+1$ if D_i (resp., Q_j) is k-ary. Informally, $\mathsf{PosD}_i(\vec{x}, y)$ asserts that the truth value should be copied to the y-th cell of the input tape; the meaning of PosQ_j is defined similarly. Since it is a folklore in the KR community as how to define such relations by arithmetic relations, we omit the details here.

The DEDs below are designed to copy the database:

$$\mathsf{D}_i(\vec{y}), \mathsf{PosD}_i(\vec{y}, z), \mathsf{BCQ}(x) \to \mathsf{ITape}_{\bar{1}}(z, x) \tag{21}$$
$$\mathsf{PosD}_i(\vec{y}, z), \mathsf{BCQ}(x) \to \mathsf{ITape}_{\bar{0}}(z, x) \tag{22}$$

At a first glance, the above encodings seem problematic since $\mathsf{ITape}_{\bar{0}}(z, x)$ and $\mathsf{ITape}_{\bar{1}}(z, x)$ might be true simultaneously. However, by the convergence of M and the analysis in Subsection 4.1 we know that this does not lead to any trouble.

Next, let us fix the way to encode BCQs (e.g., the Gödel numbering). By the arithmetic relations, we can define relations BCQ, HasQ_i and NHasQ_i for all proper indices i such that $\mathsf{BCQ}(x)$ asserts that x encodes some BCQ over $\mathcal{Q}$, and $\mathsf{HasQ}_i(\vec{x}, y)$ (resp., NHasQ_i) asserts that $\mathsf{Q}_i(\vec{x})$ appears (resp., does not appear) in the BCQ encoded by y. To copy the focused BCQ, we use the following DEDs:

$$\mathsf{HasQ}_i(\vec{y}, x), \mathsf{PosQ}_i(\vec{y}, z), \mathsf{BCQ}(x) \to \mathsf{ITape}_1(z, x) \tag{23}$$
$$\mathsf{NHasQ}_i(\vec{y}, x), \mathsf{PosQ}_i(\vec{y}, z), \mathsf{BCQ}(x) \to \mathsf{ITape}_0(z, x) \tag{24}$$

Transition. Simulating transitions of M is almost the same as that in the classical case, see, e.g., [Dantsin *et al.*, 2001; Baget *et al.*, 2011]. The only difference here is that we have to treat nondeterminism. It should be noted that this cannot be directly encoded by disjunctions. However, we can first guess a number which encodes the action choices in all transitions, and then simulate the computation on these choices, which is deterministic and can be encoded in a routine way. Note that the "guess" will be implemented in the next part.

To carry out the idea, as mentioned in the begining of this subsection, for each relation contributing to defining the computation path, we use an additional argument to specify (the number encoding) the choices. Without loss of generality, let $\delta(s, c) = \{(s_0, c_0, d_0), (s_1, c_1, d_1)\}$ be a transition of M. Let $\phi_i(x, \vec{y}) \rightarrow \psi_i(\vec{z})$ be a DED to simulate the deterministic transition $\delta(s, c) = (s_i, c_i, d_i)$ in the x-th stage of the computation. Now, our transition can be simulated by

$$\mathsf{Bit}_i(v, x), \phi_i'(x, \vec{y}, v) \rightarrow \psi_i'(\vec{z}, v) \tag{25}$$

for all $i \in \{0, 1\}$, where ϕ_i' and ψ_i' are obtained from ϕ_i and ψ_i, respectively, by adding the argument v to related atoms. Intuitively, these DEDs assert that M will perform the first action if the x-bit of v is 0, otherwise M will carry out the second one.

Acceptance. Since M is nondeterministic, to describe that M accepts an input, we have to state that there is an accepting state that can be reached from the initial state via a sequence of action choices (encoded by a number). To implement this, for each accepting state s, we construct a DED as follows:

$$\mathsf{BCQ}(z), \mathsf{State}_s(x, y, z) \rightarrow \mathsf{Accept}(z) \tag{26}$$

Finally, let Σ_{sim} consist of all DEDs defined in this subsection. Then we have an encoding to simulate machine M.

Proposition 13. *Let D be a database over $\mathcal{D}$, and I be a minimal model of $D \cup \Sigma_{num} \cup \Sigma_{sim}$ such that* $\mathsf{Undesired} \notin I$. *Then, for all $a \in adom(I)$, $\mathsf{Accept}(a) \in I$ iff a encodes a BCQ $\boldsymbol{q}$ over $\mathcal{Q}$ (in the fixed way) such that $(D, \boldsymbol{q}) \in O$.*

5.4 Generating universal model

Now it remains to show how to generate the universal model $U_O(D)$. Let a be a number that encodes a BCQ $\boldsymbol{q}$ such that $\mathsf{Accept}(a)$ is true in the intended instance. For all atoms α over $\mathcal{Q}$, we first test whether α appears in $\boldsymbol{q}$. If the answer is yes we then copy α to the universal model. Since $U_O(D)$ is defined by a disjoint union of $[\boldsymbol{q}]$, a renaming of variables in $\boldsymbol{q}$ would be necessary. As variables are encoded by numbers, the renaming can be carried out by mapping a number to another number that is big enough. It is easy to see that the desired mapping can be defined by arithmetic relations.

With this assumption, we introduce two fresh relation symbols QVar and NewV where $\mathsf{QVar}(y, x)$ asserts that y is an existential variable which appears in the BCQ encoded by x, and $\mathsf{NewV}(y, z, x)$ asserts that the variable y which appears in the BCQ encoded by x will be mapped to z. Now, we use them to define a new relation Name, where $\mathsf{Name}(y, z, x)$ means that y will be replaced with z in the copy of the BCQ encoded by x. Below are the DEDs that implement it.

$$\mathsf{BCQ}(x), \mathsf{QVar}(y, x), \mathsf{NewV}(y, z, x) \rightarrow \mathsf{Name}(y, z, x) \tag{27}$$

$$\mathsf{BCQ}(x), \mathsf{DC}(y) \rightarrow \mathsf{Name}(y, y, x) \tag{28}$$

where the second one means that we do not change constants.

Next, to copy all the atoms that appear in the BCQ to the intended universal model, we employ the following DEDs:

$$\mathsf{Accept}(x), \mathsf{HasQ}_i(\vec{y}, x), \mathsf{Name}(\vec{y}, \vec{z}, x) \rightarrow \mathsf{Q}_i(\vec{z}) \tag{29}$$

where $\mathsf{Name}(\vec{y}, \vec{z}, x)$ denotes $\bigwedge_{1 \leq j \leq k} \mathsf{Name}(y_j, z_j, x)$ if $\vec{y} = y_1 \cdots y_k$, $\vec{z} = z_1 \cdots z_k$, and k is the arity of Q_i.

In addition, let us focus on an instance which defines a bad successor relation. To avoid changing the semantics, we have to force the instance to accept all BCQs, which can be done by defining the following DED for each k-ary Q_i:

$$\mathsf{Undesired}, \mathsf{DC}(x_1), \ldots, \mathsf{DC}(x_k) \rightarrow \mathsf{Q}_i(x_1, \ldots, x_k) \tag{30}$$

Finally, let Σ_{um} denote the set of all DEDs defined here.

Proposition 14. *Let D be a database over $\mathcal{D}$, and let I be a minimal model of $D \cup \Sigma_{num} \cup \Sigma_{sim} \cup \Sigma_{um}$ such that* $\mathsf{Undesired} \notin I$. *Then $I|_{\mathcal{Q}}$ is isomorphic to $U_O(D)$.*

Now, by Propositions 11 and 14 we have Theorem 7.

6 Related work and discussions

Rudolph and Thomazo [2015] identified an elegant characterization for the expressiveness of TGDs, which asserts that TGDs define all the recursively enumerable database queries which are closed under homomorphisms. However, TGDs there are treated as a classical database query language. Our work focuses on the expressiveness of DEDs as a language for ontology-based query answering. As an immediate consequence, we have also established a characterization for the expressiveness of DEDs as a classical query language.

Gottlob *et al.* [2014a] proved that an ontology expressed by a finite set of TGDs and negative constraints (NCs) admits first-order rewritings if, and only if, it can be expressed by a finite set of TGDs and NCs that enjoys the bounded derivation depth property, which can be regarded as a relative expressive completeness w.r.t. a semantic class of existential rules. It would be interesting to know whether their characterization can be generalized to arbitrary OCQA ontologies. Zhang *et al.* [2015] showed that every TGD set with finite Skolem chase can be rewritten as a weakly-acyclic one, which is known as another relatively expressive completeness result.

It is also worth mentioning some other work related to ours. Gottlob *et al.* [2014b] showed that, as a classical query language, the weakly-guarded TGDs with stratified negations capture the class of queries decidable in EXPTIME. Arenas *et al.* [2014] proposed an interesting notion called *program expressive power*, and used it to compare the expressiveness of several ontological languages. Our framework for OCQA ontology can be regarded as a significant refinement of theirs.

The idea presented in this paper may be applied to other languages. For example, we can use it to prove that TGDs capture all recursively enumerable OCQA ontologies that are closed under database homomorphisms. The only additional difficulty is how to define a linear order by TGDs, which can be achieved by using some techniques from [Rudolph and Thomazo, 2015; Gottlob *et al.*, 2014b]. Our work may shed a new light on identifying decidable languages for OCQA, and in particular on identifying an expressively complete language for first-order rewritable or tractable OCQA if it exists.

Acknowledgements

We would like to thank anonymous referees for their helpful comments and suggestions for improving the paper. The first author's research was partially supported by the National Natural Science Foundation of China under grants 61173170, 61300222, 61370230, 61433006, 61572221 and U1401258.

References

[Alviano *et al.*, 2012] Mario Alviano, Wolfgang Faber, Nicola Leone, and Marco Manna. Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues. *Theor. Pract. Log. Prog.*, 12(4-5):701–718, 2012.

[Arenas *et al.*, 2014] Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS-14)*, pages 14–26, 2014.

[Baader *et al.*, 2005] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 364–369, 2005.

[Baget *et al.*, 2011] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[Bourhis *et al.*, 2013] Pierre Bourhis, Michael Morak, and Andreas Pieris. The impact of disjunction on query answering under guarded-based existential rules. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-13)*, pages 796–802, 2013.

[Bourhis *et al.*, 2015] Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable highly expressive query languages - IJCAI-15 distinguished paper (honorary mention). In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 2826–2832, 2015.

[Calì *et al.*, 2012] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

[Calvanese *et al.*, 2007] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and expressivecient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[Dantsin *et al.*, 2001] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[Ebbinghaus *et al.*, 1994] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical logic (2. ed.)*. Undergraduate texts in mathematics. Springer, 1994.

[Fagin *et al.*, 2005] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.

[Gottlob *et al.*, 2014a] G. Gottlob, S. Kikot, R. Kontchakov, V. Podolskii, T. Schwentick, and M. Zakharyaschev. The price of query rewriting in ontology-based data access. *Artif. Intell.*, 213:42–59, 2014.

[Gottlob *et al.*, 2014b] Georg Gottlob, Sebastian Rudolph, and Mantas Simkus. Expressiveness of guarded existential rule languages. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS-14)*, pages 27–38, 2014.

[Harel, 1986] David Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *J. ACM*, 33(1):224–248, 1986.

[Immerman, 1999] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.

[Rudolph and Thomazo, 2015] Sebastian Rudolph and Michaël Thomazo. Characterization of the expressivity of existential rule queries. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 3193–3199, 2015.

[Vardi, 1982] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC-82)*, pages 137–146, 1982.

[Zhang *et al.*, 2015] Heng Zhang, Yan Zhang, and Jia-Huai You. Existential rule languages with finite chase: Complexity and expressiveness. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI-15)*, pages 1678–1685, 2015.

[Zhang *et al.*, 2016] Heng Zhang, Yan Zhang, and Jia-Huai You. Expressive completeness of existential rule languages for ontology-based query answering. *CoRR (arXiv) abs/1604.05006*, 2016.