Characterizing the Program Expressive Power of Existential Rule Languages

Heng Zhang[1], **Guifei Jiang**[2]

[1] College of Intelligence and Computing, Tianjin University, China
[2] College of Software, Nankai University, China
heng.zhang@tju.edu.cn, g.jiang@nankai.edu.cn

Abstract

Existential rule languages are a family of ontology languages that have been widely used in ontology-mediated query answering (OMQA). However, for most of them, the expressive power of representing domain knowledge for OMQA, known as the program expressive power, is not well-understood yet. In this paper, we establish a number of novel characterizations for the program expressive power of several important existential rule languages, including tuple-generating dependencies (TGDs), linear TGDs, as well as disjunctive TGDs. The characterizations employ natural model-theoretic properties, and automata-theoretic properties sometimes, which thus provide powerful tools for identifying the definability of domain knowledge for OMQA in these languages.

Introduction

Existential rule languages, a.k.a. Datalog$\pm$, had been initially introduced in databases as dependency languages to specify the semantics of data stored in a database (Abiteboul, Hull, and Vianu 1995). As one of the most popular dependency languages, tuple-generating dependencies (TGDs) and its extensions, including (disjunctive) embedded dependencies and disjunctive TGDs, had been extensively studied. Recently, these languages have been rediscovered as languages for data exchange (Fagin et al. 2005), data integration (Lenzerini 2002), ontology reasoning (Calì et al. 2010) and knowledge graph (Bellomarini et al. 2017).

A major computational task based on existential rule languages is known as ontology-mediated query answering (OMQA), which generalizes the traditional database querying by enriching database with a domain ontology. Unfortunately, even for TGDs, the problem of OMQA was proved to be undecidable (Beeri and Vardi 1981). Towards efficient reasoning, many decidable sublanguages have been identified, including linear TGDs and guarded TGDs (Calì, Gottlob, and Lukasiewicz 2012), frontier-guarded TGDs (Baget et al. 2011), sticky TGDs (Calì, Gottlob, and Pieris 2012), weakly-acyclic TGDs (Fagin et al. 2005) and shy programs (Leone et al. 2012). With these languages, it is thus important to identify their expressive power so that, given an application, we know which language should be used.

In OMQA, there have been mainly two lines of research on the language expressive power. The first line of research regards every ontology together with a classical query as a database query, usually called an ontology-mediated query. The main goal of this line is to understand which class of databases can be defined by an ontology-mediated query. We call such kind of expressive power the *data expressive power*. In contrast, the second line is concerned with which kind of domain knowledge can be expressed in an ontology language; or more formally, which classes of database-query pairs are definable in the language. Expressive power of this kind is known as the *program expressive power*, which was first proposed by Arenas, Gottlob, and Pieris (2014).

A number of papers are devoted to characterizing data expressive power of existential rule languages. An incomplete list is as follows: Gottlob, Rudolph, and Simkus (2014) proved that weakly (frontier-)guarded TGD queries with stratified negations capture the class of EXPTIME-queries; nearly (frontier-)guarded TGD queries have the same expressive power as Datalog. Rudolph and Thomazo (2015) showed that TGD queries capture the class of recursively enumerable queries closed under homomorphisms. Krötzsch and Rudolph (2011) identified that jointly acyclic TGD queries have the same expressive power as Datalog, which was later extended to TGD queries with terminating Skolem chase in (Zhang, Zhang, and You 2015). In description logics, Bienvenu et al. (2014) characterized the data expressive power of $\mathcal{ALC}$ and its variants by some interesting complexity classes and fragments of disjunctive Datalog.

Unlike the data expressive power, the program expressive power of existential rule languages is not well-understood yet. Arenas, Gottlob, and Pieris (2014) proved that Datalog is strictly less expressive than warded Datalog$^{\exists}$, and obtained a similar separation for the variants with stratified negations and negative constraints. Zhang, Zhang, and You (2016) proposed a semantic definition for ontologies in OMQA, and proved that disjunctive embedded dependencies (DEDs) capture the class of recursively enumerable OMQA-ontologies. In addition, it is implicit in (Zhang, Zhang, and You 2015) that the weakly-acyclic TGDs have the same program expressive power as all its extensions with terminating Skolem chase. This paper continues this line of work and aims at characterizing the program expressive power of several important languages including TGDs, disjunctive TGDs and linear TGDs.

Our contributions in this paper are threefold. Firstly, we show that the equalities in a finite set of DEDs are removable if, and only if, the OMQA-ontology defined by these DEDs is closed under both database homomorphisms and constant substitutions. Secondly, we prove that, under CQ-answering, every finite set of DTGDs can be translated to an equivalent finite set of TGDs, while the translatability under UCQ-answering is captured by a property called query constructivity. Thirdly, we characterize the linear TGD-definability of OMQA-ontologies by data constructivity and the recognizability of queries by a natural class of tree automata.

Preliminaries

Databases and Instances We use a countably infinite set Δ (resp., Δ_{n} and Δ_{v}) of *constants* (resp., *(labeled) nulls* and *variables*), and assume they are pairwise disjoint. Every *term* is a constant, a null or a variable. A *(relational) schema* $\mathscr{S}$ is a set of *relation symbols*, each associated a natural number called the *arity*. Every $\mathscr{S}$*-atom* is either an equality or a *relational atom* built upon terms and a relation symbol in $\mathscr{S}$. A *fact* is a variable-free relational atom, and an $\mathscr{S}$*-instance* is a set of $\mathscr{S}$-facts. A *database* is a finite instance in which no null occurs. Given an instance I, let $adom(I)$ (resp., $term(I)$) denote the set of constants (resp., terms) occurring in I. Given a set A of terms, let $I|_A$ be the maximum subset J of I such that $term(J) \subseteq A$.

Let I and J be $\mathscr{S}$-instances, and C a set of constants. A C*-homomorphism* from I to J is a function $h : adom(I) \to adom(J)$ such that $h(I) \subseteq J$ and $h(c) = c$ for all constants $c \in C$. If such h exists, we say I is C*-homomorphic* to J, and write $I \to_C J$. In addition, we write $I \rightarrowtail_C J$ if h is injective. We say I is C*-isomorphic* to J if there is a bijective C-homomorphism h from I to J such that $h(I) = J$. For simplicity, in the above, C could be dropped if it is empty. A *substitution* is a partial function from Δ_{v} to $\Delta \cup \Delta_{\mathrm{n}}$.

Queries Fix $\mathscr{S}$ as a schema. Every $\mathscr{S}$*-CQ* is a first-order formula of the form $\exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$ where $\varphi(\boldsymbol{x}, \boldsymbol{y})$ is a finite but nonempty conjunction of relational $\mathscr{S}$-atoms. An $\mathscr{S}$*-UCQ* is a first-order formula built upon $\mathscr{S}$-atoms by using connectives $\wedge, \vee$ and quantifier $\exists$ only. Clearly, every UCQ is equivalent to a disjunction of CQs, and every CQ is also a UCQ. Note that constants are allowed to appear in a query. Given a query (CQ or UCQ) q, let $const(q)$ denote the set of all constants that occur in q.

A UCQ is called *Boolean* if it has no free variables. Let BCQ be short for Boolean CQ. Given a BCQ q, let $[q]$ denote a database that consists of all atoms in q where each variable is regarded as a null. In this paper, unless otherwise stated, we only consider Boolean queries. Let CQ (resp., UCQ) denote the class of Boolean CQs (resp., Boolean UCQs).

Existential Rule Languages Let $\mathscr{S}$ be a schema. Then every *disjunctive embedded dependency (DED)* over $\mathscr{S}$ is a first-order sentence σ of the form

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

where $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ are tuples of variables, ϕ a conjunction of relational $\mathscr{S}$-atoms involving terms only from $\boldsymbol{x} \cup \boldsymbol{y}$, each ψ_i a conjunction of $\mathscr{S}$-atoms involving terms only from $\boldsymbol{x} \cup \boldsymbol{z}$, and every variable in $\boldsymbol{x}$ has at least one occurrence in ϕ. For simplicity, we omit universal quantifiers and brackets outside the atoms. Let $head(\sigma) = \{\psi_i : 1 \leq i \leq k\}$ and $body(\sigma) = \phi$, called the *head* and *body* of σ, respectively.

Disjunctive tuple-generating dependencies (DTGDs) are defined as equality-free DEDs, and *tuple-generating dependencies (TGDs)* are disjunction-free DTGDs. A TGD is called *linear* if its body consists of a single atom. A DED of the form (1) is *canonical* if $\psi_i, 1 \leq i \leq k$, consists of a single atom. It is well-known that, by introducing auxiliary relation symbols, every set of DEDs (resp, DTGDs, TGDs and linear TGDs) can be converted to an equivalent (under query answering) set of canonical DEDs (resp., DTGDs, TGDs and linear TGDs). Hence, unless stated otherwise, we assume *dependencies are canonical* in the rest of this paper.

Let D be a database, Σ a set of DEDs, and q a Boolean UCQ. We write $D \cup \Sigma \vDash q$ if, for all instances I, if $D \subseteq I$ and I is a model of σ for all $\sigma \in \Sigma$, then I is also a model of q, where the notion of *model* is defined in a standard way.

OMQA-ontologies In this subsection, we introduce some notions related to OMQA-ontology. For more details, please refer to (Zhang, Zhang, and You 2016). Let $\mathscr{D}$ and $\mathscr{Q}$ be a disjoint pair of schemas, and $\mathcal{Q}$ a class of Boolean UCQs. Every *quasi-OMQA*$[\mathcal{Q}]$*-ontology* over $(\mathscr{D}, \mathscr{Q})$ is a set of database-query pairs (D, q), where D is a nonempty $\mathscr{D}$-database and q a Boolean $\mathscr{Q}$-UCQ in $\mathcal{Q}$ such that $const(q) \subseteq adom(D)$. Furthermore, an *OMQA*$[\mathcal{Q}]$*-ontology* is a quasi-OMQA$[\mathcal{Q}]$-ontology O that admits the following properties:

1. (*Closure under Query Conjunctions*) If $p \wedge q \in \mathcal{Q}$, $(D, p) \in O$ and $(D, q) \in O$, then $(D, p \wedge q) \in O$;
2. (*Closure under Query Implications*) If $p \in \mathcal{Q}$, $q \vDash p$ and $(D, q) \in O$, then $(D, p) \in O$;
3. (*Closure under Injective Database Homomorphisms*) If $(D, q) \in O$ and $D \rightarrowtail_{const(q)} D'$, then $(D', q) \in O$;
4. (*Closure under Constant Renaming*) If $(D, q) \in O$ and τ is a *constant renaming* (i.e., a partial injective function from Δ to Δ), then $(\tau(D), \tau(q)) \in O$.

Given a set Σ of DEDs, let $[\![\Sigma]\!]^{\mathcal{Q}}_{\mathscr{D},\mathscr{Q}}$ denote the set of all database-query pairs (D, q) where D is a $\mathscr{D}$-database, $q \in \mathcal{Q}$ a $\mathscr{Q}$-UCQ, and $D \cup \Sigma \vDash q$. Given an OMQA$[\mathcal{Q}]$-ontology O over $(\mathscr{D}, \mathscr{Q})$, we say O is *defined by* Σ if $O = [\![\Sigma]\!]^{\mathcal{Q}}_{\mathscr{D},\mathscr{Q}}$.

The following characterization for DEDs is actually implicit in (Zhang, Zhang, and You 2016; Zhang et al. 2020).

Theorem 1 (Zhang, Zhang, and You)**.** *An OMQA*[UCQ]*-ontology is defined by a finite set of DEDs iff it is recursively enumerable.*

For convenience, given a class $\mathcal{Q}$ of Boolean UCQs, every *DED*$[\mathcal{Q}]$*-ontology* (resp., *DTGD*$[\mathcal{Q}]$*-ontology* and *TGD*$[\mathcal{Q}]$*-ontology*) is defined as an OMQA$[\mathcal{Q}]$-ontology which is defined by some finite set of DEDs (resp., DTGDs and TGDs).

DTGDs

In this section, we examine the program expressive power of DTGDs. To do it, we first present a novel chase algorithm for DTGDs, which also plays a key role in the next section.

Nondeterministic Chase Let $\mathscr{D}$ be a schema. A *nondeterministic fact* (over $\mathscr{D}$) is a finite disjunction of ($\mathscr{D}$-)facts. For convenience, we often regard each nondeterministic fact as a set of ground atoms. Every *nondeterministic instance* (over $\mathscr{D}$) is defined as a set of nondeterministic facts (over $\mathscr{D}$).

Let I be a nondeterministic instance, and σ a DTGD in which $\alpha_1, \ldots, \alpha_n$ list all the atoms in the body. We say σ is *applicable to* I if there is a substitution h and a tuple $\boldsymbol{F}$ of nondeterministic facts $F_1, \ldots, F_n \in I$ such that $h(\alpha_i) \in F_i$ for all $i = 1, \ldots, n$. In this case, we let $res(\boldsymbol{F}, \sigma, h)$ denote the nondeterministic fact defined as follows:

$$h'(\mathit{head}(\sigma)) \cup \bigcup_{i=1}^{n} F_i \setminus \{h(\alpha_i)\}$$

where h' is a substitution that extends h by mapping each existential variable v in σ to a null which one-one corresponds to the triple $(\sigma, h(\boldsymbol{x}), v)$, and $\boldsymbol{x}$ denotes the tuple of variables occurring in both the head and the body of σ. In addition, we call $res(\boldsymbol{F}, \sigma, h)$ a *result of applying* σ *to* I.

Furthermore, given a database D and a set Σ of DTGDs, let $chase_0(D, \Sigma) = D$; for $k > 0$ let $chase_k(D, \Sigma)$ denote the union of $chase_{k-1}(D, \Sigma)$ and the set of all results of applying σ to $chase_{k-1}(D, \Sigma)$ for all $\sigma \in \Sigma$. Let $chase(D, \Sigma)$ denote the union of $chase_k(D, \Sigma)$ for all $k \geq 0$.

In above definitions, if Σ is a set of TGDs, the procedure of nondeterministic chase will degenerate into the traditional oblivious Skolem chase, see, e.g., (Marnette 2009).

The following theorem gives the soundness and completeness of the nondeterministic chase.

Theorem 2. *Let Σ be a set of DTGDs, D be a database, and q be a Boolean UCQ. Then $D \cup \Sigma \models q$ iff $chase(D, \Sigma) \models q$, where by the notation $chase(D, \Sigma) \models q$ we denote that q is a logical consequence of $chase(D, \Sigma)$ as usual.*

Now we generalize the notion of homomorphism from instances to nondeterministic instances. Let I and J be nondeterministic instances over the same schema. Given a set C of constants, a function $h : adom(I) \to adom(J)$ is called a C-*homomorphism* from I to J, written $h : I \to_C J$, if we have $h(I) \subseteq J$ and $h(c) = c$ for all constants $c \in C$.

The proposition below shows that the nondeterministic chase preserves the generalized homomorphisms. This property will play an important role in our first characterization.

Proposition 3. *Let Σ be a set of DTGDs, let D and D' be databases, and let C be a set of constants. If there exists a C-homomorphism τ from D to D', then there exists a C-homomorphism $\tau' \supseteq \tau$ from $chase(D, \Sigma)$ to $chase(D', \Sigma)$.*

Characterization In this subsection, we establish a characterization for DTGDs. Before proceeding, we need to present some properties for OMQA-ontologies.

Let $\mathcal{Q}$ be a class of UCQs. An OMQA[$\mathcal{Q}$]-ontology O is said to be *closed under database homomorphisms* if, for all $(D, q) \in O$, if D' is a database with $D \to_{\mathit{const}(q)} D'$, then $(D', q) \in O$; and O is *closed under constant substitutions* if, for all $(D, q) \in O$, if τ is a *constant substitution* (i.e., a partial function from Δ to Δ), then $(\tau(D), \tau(q)) \in O$.

The following proposition tell us that ontologies defined by DTGDs are closed under both of above properties.

Lemma 4. *Every DTGD*[UCQ]*-ontology is closed under both database homomorphisms and constant substitutions.*

Moreover, we can show that the properties above exactly capture the class of DED-ontologies definable by DTGDs.

Theorem 5. *A DED*[UCQ]*-ontology is defined by a finite set of DTGDs iff it is closed under both database homomorphisms and constant substitutions.*

Sketch of Proof. The direction of "only-if" immediately follows from Lemma 4. It thus remains to consider the converse. Let O be a DED[UCQ]-ontology closed under both database homomorphisms and constant substitutions, and let Σ be a finite set of DEDs that defines O. We need to construct a finite set Σ' of DTGDs which plays the same role as Σ under the semantics of UCQ-answering.

To implement the construction, we introduce *Eq* as a fresh binary relation symbol, and use some DTGDs to assert that *Eq* defines an equivalence relation. For each (k-ary) relation symbol R occurring in Σ, let the following DTGD

$$\wedge_{i=1}^{k} \mathit{Eq}(x_i, y_i) \wedge R(x_1, \ldots, x_k) \to R(y_1, \ldots, y_k) \quad (2)$$

to assure that all the terms (constants or nulls) equivalent w.r.t. *Eq* will play the same role in R.

Moreover, we simulate each DED $\sigma \in \Sigma$ by a DTGD σ^*, which is obtained from σ by substituting *Eq* for every occurrence of the equality symbol $=$. Let Σ' be the set consisting of all the DTGDs mentioned above. Thanks to the closure of O under both database homomorphisms and constant substitutions, one can prove that the transformation preserves the semantics of UCQ-answering, i.e., $D \cup \Sigma \models q$ iff $D \cup \Sigma' \models q$ for all $\mathscr{D}$-databases D and Boolean $\mathscr{Q}$-UCQs q. Thus, Σ' is the desired DTGD set, which completes the proof. $\square$

Let UCQ^- denote the class of all Boolean UCQs involving no constant. For query answering with queries in UCQ^-, the above characterization can be simplified as follows:

Corollary 6. *A DED*[UCQ^-]*-ontology is defined by a finite set of DTGDs iff it is closed under database homomorphisms.*

TGDs

In this section, let us consider another important existential rule language, TGDs, a sublanguage of DTGDs in which disjunctions are not allowed to appear in the rule head.

Characterization for CQ-answering We first show that, in the case of CQ-answering, disjunctions can be removed from DTGDs. In other words, TGDs have the same expressive power as DTGDs under CQ-answering.

Theorem 7. *Every DTGD*[CQ]*-ontology is defined by a finite set of TGDs.*

To prove this theorem, it suffices to translate every set of DTGDs to a set of TGDs such that they define the same ontology under CQ-answering. Suppose O is a DTGD[CQ]-ontology over a schema pair $(\mathscr{D}, \mathscr{Q})$, and Σ a set of canonical DTGDs that defines O. The general idea is to construct a set Σ^* of TGDs such that the deterministic chase on Σ^*

simulates the nondeterministic chase on Σ. The desired simulation employs a technique used in Section 3 of (Zhang and Zhang 2017) in which the progression of disjunctive logic programs is simulated by normal logic programs. The main difficulty here is that we need to treat CQ-answering.

To encode a nondeterministic fact, we need a set of numbers and an encoding function. The encoding function is defined by a ternary relation symbol *Enc*. By $Enc(x, y, z)$ we mean that z encodes the pair (x, y). Numbers used in the encoding are collected by a unary relation symbol *Num*. Note that numbers here are not necessary to be natural numbers. For a technical reason, we also use a unary relation symbol *GT* to collect the set of all ground terms that would be used. Next, we show how to implement the encoding.

For every relation symbol $R \in \mathscr{D}$, we introduce the TGDs

$$R(x_1, \ldots, x_k) \;\to\; \wedge_{i=1}^{k} \left(Num(x_i) \wedge GT(x_i) \right) \quad (3)$$

$$\to\; \exists x\, Flag_R(x) \wedge Num(x) \quad (4)$$

where k is the arity of R, and $Flag_R$ a unary relation symbol that defines a flag for the relation R. The first TGD asserts that all parameters of R are both numbers and ground terms, and the second one asserts that the flag for R must exist and, in particular, it is also a number.

To define the encoding function, we use the TGD

$$Num(x) \wedge Num(y) \to \exists z\, Enc(x, y, z) \wedge Num(z) \quad (5)$$

which asserts that, for all numbers x and y, there is a number z to encode the pair (x, y). With the relations defined above, we are then able to encode (ground) atoms. For example, to encode the atom $\alpha = R(x_1, x_2)$, we use the formula

$$Flag_R(y_1) \wedge Enc(y_1, x_1, y_2) \wedge Enc(y_2, x_2, y_3)$$

which asserts that y_3 is a number encoding the atom α. Note that α is regarded as the triple (y_1, x_1, x_2) where y_1 is the flag of R, denoting where the encoding of the first element of the tuple is. In addition, to simplify the notation, given a formula $\varphi(z_0, \boldsymbol{z})$, we often use $\varphi(\lceil \alpha \rceil, \boldsymbol{z})$ to denote

$$Flag_R(y_1) \wedge Enc(y_1, x_1, y_2) \wedge Enc(y_2, x_2, y_3) \wedge \varphi(y_3, \boldsymbol{z}). \quad (6)$$

To encode a disjunction (resp., conjunction) of formulas, we need a flag to denote where the encoding of the first disjunct (resp., conjunct) is. To generate such flags, we use

$$\to\; \exists x\, Flag_d(x) \wedge Num(x) \quad (7)$$

$$\to\; \exists x\, Flag_c(x) \wedge Num(x) \quad (8)$$

where $Flag_d$ (resp., $Flag_c$) is a unary relation symbol intended to define the flag of encoding disjunction (resp., conjunction). The way of encoding a disjunction (conjunction) is similar to that for atoms, but with a different flag. In addition, the notation $\lceil \cdot \rceil$ can also be extended to disjunctions and conjunctions in an obvious way.

With the above relations, we are able to encode nondeterministic facts. To access nondeterministic facts, some relations are needed. We introduce fresh relation symbols *NF*, *Mrg* and *Eq*. By $NF(x)$ we mean that x encodes a nondeterministic fact. By $Mrg(x, y, z)$ we denote that z encodes a disjunction (which is a nondeterministic fact) of the nondeterministic facts encoded by x and y. Moreover, $Eq(x, y)$ asserts that the nondeterministic facts encoded by x and y are equivalent, i.e., they consist of the same set of ground atoms. We only show how to define the merging operation:

$$NF(x) \wedge Flag_d(y) \to Mrg(x, y, x) \quad (9)$$

$$Mrg(x, u, v) \wedge Enc(u, w, y) \wedge Enc(v, w, z) \to Mrg(x, y, z) \quad (10)$$

To simplify the notation, let $Mrg(t_1, \ldots, t_k; x_k)$ be short for

$$Flag_d(x_0) \wedge Mrg(t_1, x_0, x_1) \wedge \cdots \wedge Mrg(t_k, x_{k-1}, x_k).$$

Next let us construct TGDs to simulate the nondeterministic chase on Σ. We introduce *True* as a fresh unary relation symbol, and by $True(x)$ we mean that the formula encoded by x can be inferred from the set of nondeterministic facts generated by the chase. For each canonical DTGD $\sigma \in \Sigma$, if $\alpha_1, \ldots, \alpha_k$ list all atoms in the body of σ, we use the following TGDs to simulate the nondeterministic chase for σ:

$$\begin{aligned} &\wedge_{i=1}^{k} (NF(v_i) \wedge Enc(u_i, \lceil \alpha_i \rceil, v_i) \wedge True(v_i)) \\ &\quad \wedge Mrg(u_1, \ldots, u_k; y) \to \exists \boldsymbol{z}\, T_\sigma(\boldsymbol{x}, y, \boldsymbol{z}) \wedge Num(\boldsymbol{z}) \end{aligned} \quad (11)$$

$$T_\sigma(\boldsymbol{x}, y, \boldsymbol{z}) \wedge Mrg(y, \lceil head(\sigma) \rceil, w) \to True(w) \quad (12)$$

To initialize the truth of relations over the data schema $\mathscr{D}$, for each k-ary $R \in \mathscr{D}$, we introduce the following TGD:

$$\begin{aligned} &R(x_1, \ldots, x_k) \wedge Flag_R(y_0) \wedge Enc(y_0, x_1, y_1) \wedge \cdots \\ &\quad \wedge Enc(y_{k-1}, x_k, y_k) \to True(y_k) \end{aligned} \quad (13)$$

To make sure that the equivalent facts play the same role in the chase procedure, we define the following TGD:

$$NF(x) \wedge NF(y) \wedge True(x) \wedge Eq(x, y) \to True(y) \quad (14)$$

Let Σ' denote the set of all TGDs defined above. Fix a database D. By definition, it is easy to see that symbol *Enc* defines an encoding function in $chase(D, \Sigma')$. That is, for all numbers a, b defined by *Num* in $chase(D, \Sigma')$, there is exactly one term c such that $Enc(a, b, c)$ holds in $chase(D, \Sigma')$. Moreover, each symbol in $Flag_R, Flag_d, Flag_c$ defines exactly one number (called a flag) in $chase(D, \Sigma')$. Given a nondeterministic fact F, let $\langle F \rangle$ denote the number encoding F under the defined encoding function and flags. By an induction on chase, one can prove the following:

Lemma 8. $F \in chase(D, \Sigma)$ *iff* $True(\langle F \rangle) \in chase(D, \Sigma')$.

With this lemma, to construct the desired TGD set Σ^*, it remains to define some TGDs which generate the BCQs derivable from $chase(D, \Sigma)$. The following property will play an important role in implementing this task.

Lemma 9. *Let Σ be a finite set of DTGDs, D a database, and q a BCQ of the form $\exists \boldsymbol{x} \varphi(\boldsymbol{x})$ where φ is quantifier-free and $\boldsymbol{x}$ is a tuple of length k which lists all the free variables in φ. Then $D \cup \Sigma \vDash q$ iff there exists a finite set $T \subseteq term(chase(D, \Sigma))^k$ such that $chase(D, \Sigma) \vDash \bigvee_{\boldsymbol{t} \in T} \varphi(\boldsymbol{t})$.*

To implement the above idea, we need more relation symbols, including *DNF* and *Normalize*. By $DNF(x)$ we denote that the (quantifier-free) formula encoded by x is of disjunctive normal form (DNF), and by $Normalize(x, y, z)$ we mean that z encodes a DNF-formula obtained from the conjunction of (DNF-formulas encoded by) x and y by applying the distributive law. Such relations can be defined in TGDs by recursions in a routine way. We omit the details here.

To encode BCQs, we need to generate an infinite number of variables, which can be done by the following TGDs:

$$\rightarrow \quad \exists x\, \mathit{Var}(x) \wedge \mathit{Num}(x) \tag{15}$$

$$\mathit{Var}(x) \quad \rightarrow \quad \exists y\, \mathit{Next}(x,y) \wedge \mathit{Var}(y) \wedge \mathit{Num}(y) \tag{16}$$

where $\mathit{Var}(x)$ asserts that x is a variable, and $\mathit{Next}(x,y)$ denotes that y is the variable immediately after x. The generated variables will be used as numbers. Furthermore, we use $\mathit{BCQ}(x)$ to denote that x encodes a BCQ. Note that all variables in a BCQ are existential, so we can omit the quantifiers, and simply regard it as a finite conjunction of atoms.

In addition, we introduce a fresh binary relation symbol *Match*. By $\mathit{Match}(x,y)$ we mean that y encodes a ground DNF-formula in which each disjunct ψ is an instantiation of the BCQ q encoded by x, that is, ψ can be obtained from q by substituting some ground term for each existential variable.

With the above relations, we are now able to generate all the numbers encoding BCQs derivable from $chase(D,\Sigma)$.

$$\mathit{True}(x) \wedge \mathit{True}(y) \wedge \mathit{Normalize}(x,y,z) \rightarrow \mathit{True}(z) \tag{17}$$

$$\mathit{BCQ}(x) \wedge \mathit{DNF}(y) \wedge \mathit{True}(y) \wedge \mathit{Match}(x,y) \rightarrow \mathit{True}(x) \tag{18}$$

To make sure that the BCQs encoded by this class of numbers are derivable from $chase(D,\Sigma^*)$, we employ Zhang *et al.*'s technique of generating universal model (see Subsection 5.4 and Proposition 11 in (Zhang, Zhang, and You 2016)). Given a class $\mathbb{K}$ of databases over the same schema and a set C of constants, let $\bigoplus_C \mathbb{K}$ denote the *C-disjoint union* of $\mathbb{K}$, that is, the instance $\bigcup\{D^* : D \in \mathbb{K}\}$ where, for every $D \in \mathbb{K}$, D^* is an isomorphic copy of D such that, for each pair of distinct databases D_1 and D_2 in $\mathbb{K}$, only constants from C will be shared by D_1^* and D_2^*.

Given an OMQA[CQ]-ontology O and a database D over a proper schema, the *universal model* of O w.r.t. D, denoted $U_O(D)$, is defined as follows:

$$U_O(D) = \bigoplus\nolimits_{adom(D)} \{[q] : (D,q) \in O\}.$$

Lemma 10 (Zhang, Zhang, and You 2016)**.** *Let O be an OMQA[CQ]-ontology O over a schema pair $(\mathscr{D},\mathscr{Q})$, D a $\mathscr{D}$-database and q a $\mathscr{Q}$-BCQ. Then $(D,q) \in O$ iff $U_O(D) \models q$.*

With the above lemma, it remains to show how to generate the universal model $U_O(D)$. Let a be a number that encodes a BCQ q such that $\mathit{True}(a)$ holds in the intended instance. For all $\mathscr{Q}$-atoms α, we first test whether α appears in 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 $[q]$, a renaming of variables in q would be necessary, which can be achieved by using existential variable in the rule head to generate nulls. We introduce a relation symbol *Ren*, and by $\mathit{Ren}(y,z,x)$ we mean that y will be replaced with z in the copy of BCQ (encoded by) x. Below are some TGDs to implement it:

$$\mathit{BCQ}(x) \wedge \mathit{Var}(y) \rightarrow \exists z\, \mathit{Ren}(y,z,x) \tag{19}$$

$$\mathit{BCQ}(x) \wedge \mathit{GT}(y) \rightarrow \mathit{Ren}(y,y,x) \tag{20}$$

where the second TGD means that all the constants appearing in the BCQ will not be changed in the copy.

To generate the universal model $U_O(D)$, we still need to introduce a relation symbol *HasQ* for each relation symbol $Q \in \mathscr{Q}$. By $\mathit{HasQ}(\boldsymbol{y},x)$ we mean that $Q(\boldsymbol{y})$ is an atom appearing in the BCQ encoded by x. By traversing the whole BCQ, it is easy to see that *HasQ* can be defined by TGDs. To copy all the atoms involving Q and appearing in the BCQ to the universal model, we employ the following TGD:

$$\mathit{BCQ}(x) \wedge \mathit{True}(x) \wedge \mathit{HasQ}(\boldsymbol{y},x) \wedge \mathit{Ren}(\boldsymbol{y},\boldsymbol{z},x) \rightarrow Q(\boldsymbol{z}) \tag{21}$$

where $\mathit{Ren}(\boldsymbol{y},\boldsymbol{z},x)$ denotes formula $\bigwedge_{1\le j\le k} \mathit{Ren}(y_j,z_j,x)$ if $\boldsymbol{y} = y_1 \cdots y_k$, $\boldsymbol{z} = z_1 \cdots z_k$, and k is the arity of Q.

Let Σ^* be the set of TGDs defined in this subsection. Then the following property holds, which yields Theorem 7.

Proposition 11. *For every pair of $\mathscr{D}$-database D and $\mathscr{Q}$-BCQ q, we have $chase(D,\Sigma) \models q$ iff $chase(D,\Sigma^*) \models q$.*

Characterization for UCQ-answering It is worth noting that the translation proposed in the last subsection does not work for UCQ-answering. In this subsection, we examine the expressive power of TGDs for this case.

We first define a property. An OMQA[UCQ]-ontology O is said to *admit query constructivity* if $(D, p \vee q) \in O$ implies either $(D,p) \in O$ or $(D,q) \in O$. The following theorem tells us that the above property exactly captures the definability of a DTGD[UCQ]-ontology by TGDs.

Theorem 12. *A DTGD[UCQ]-ontology is defined by a finite set of TGDs iff it admits query constructivity.*

To prove this theorem, we need some notation and property. Given an OMQA[UCQ]-ontology O, let $O|_{\mathsf{CQ}}$ denote $\{(D,q) \in O : q \in \mathsf{CQ}\}$ which is an OMQA[CQ]-ontology.

Lemma 13. *Let O and O' be OMQA[UCQ]-ontologies that admit query constructivity. If $O|_{\mathsf{CQ}} = O'|_{\mathsf{CQ}}$ then $O = O'$.*

Now we are in the position to prove Theorem 12.

Proof of Theorem 12. The direction of "if" follows from Lemma 13 and Theorem 7. For the converse, we assume O is defined by a finite set Σ of TGDs. Let $(D, p \vee q) \in O$, where p and q are Boolean UCQs. By the completeness of the chase procedure, it holds that $chase(D,\Sigma) \models p \vee q$. Note that $chase(D,\Sigma)$ here is a deterministic instance. We thus have either $chase(D,\Sigma) \models p$ or $chase(D,\Sigma) \models q$. By the soundness of the chase, either $(D,p) \in O$ or $(D,q) \in O$ must be true, which yields the desired direction. $\square$

Example 1. *Let $\mathscr{D}$ be the schema $\{P\}$, and $\mathscr{Q}$ be the schema $\{Q,R\}$, where P,Q and R are unary relation symbols. Let Σ be a set consisting of a single DTGD defined as follows:*

$$P(x) \rightarrow Q(x) \vee R(x) \tag{22}$$

Let $D = \{P(a)\}$. Clearly, $D \cup \Sigma \models Q(a) \vee R(a)$, but neither $D \cup \Sigma \models Q(a)$ nor $D \cup \Sigma \models R(a)$. So the ontology defined by Σ over $(\mathscr{D},\mathscr{Q})$ does not admit query constructivity.

By the above example and Theorem 12, we thus have:

Corollary 14. *There is a DTGD[UCQ]-ontology that is not defined by any finite set of TGDs.*

The next corollary immediately follows from Theorem 12 and Lemma 13. With it, to examine the expressive power of TGDs, we need only to consider CQ-answering. In the next section, we will thus focus on CQ-answering.

Corollary 15. *Let $\mathscr{D}$ and $\mathscr{Q}$ be schemas, Σ and Σ' finite sets of TGDs. Then $[\![\Sigma]\!]^{\mathsf{UCQ}}_{\mathscr{D},\mathscr{Q}} = [\![\Sigma']\!]^{\mathsf{UCQ}}_{\mathscr{D},\mathscr{Q}}$ iff $[\![\Sigma]\!]^{\mathsf{CQ}}_{\mathscr{D},\mathscr{Q}} = [\![\Sigma']\!]^{\mathsf{CQ}}_{\mathscr{D},\mathscr{Q}}$.*

Linear TGDs

In this section, we focus on the program expressive power of linear TGDs. Before establishing the characterization, we need to recall some notions and make a few assumptions.

Tree Automata First recall some notions of tree automata. For more details, please refer to, e.g., (Comon et al. 2007).

Let $\mathcal{L}$ be a nonempty set of labels. An *$\mathcal{L}$-labeled tree* T is a quadruple (V, E, r, L) where $E \subseteq V \times V$, (V, E) defines a tree with the *root* $r \in V$ in a standard way, and $L : V \to \mathcal{L}$ is called the *label function*. T is called *finite* if V is finite.

Every *ranked input alphabet* is a finite and nonempty set of *input symbols*, each is a pair $\omega = (\ell(\omega), ar(\omega))$, where $\ell(\omega)$ is the *letter* of ω, and $ar(\omega)$ a natural number called the *arity* of ω. Given a ranked input alphabet Ω, an *Ω-ranked tree* is a finite labeled tree $\mathfrak{T} = (V, E, r, L)$ over Ω such that every node $v \in V$ has exactly $ar(L(v))$ children in $\mathfrak{T}$.

For convenience, we often use expressions built over Ω to denote ranked trees. A nullary input symbol $\pi \in \Omega$ denotes a ranked tree consisting of a single node with the label π. Let $\omega \in \Omega$ be a k-ary symbol, $e_1, \ldots, e_k$ be expressions denoting Ω-ranked trees $\mathfrak{T}_1, \ldots, \mathfrak{T}_k$. We then use the expression $\omega(e_1, \ldots, e_k)$ to denote the Ω-ranked tree $\mathfrak{T}$, in which the root r is labeled as ω, such that for every $i = 1, \ldots, k$, $\mathfrak{T}_i$ is a subtree of $\mathfrak{T}$ and the i-th child of r is the root of $\mathfrak{T}_i$.

Moreover, a *nondeterministic (bottom-up) tree automaton* (NTA) $\mathcal{A}$ is defined as a quadruple (S, F, Ω, Θ) where

1. S is a finite set of *states*, and $F \subseteq S$ a set of *final states*;
2. Ω is a ranked input alphabet;
3. $\Theta \subseteq \Omega \times S^* \times S$ is a transition relation which consists of *transition rules* of the form $(\omega, (s_1, \ldots, s_k), s_0)$, where $\omega \in \Omega$ is a k-ary symbol for some k and $s_0, \ldots, s_k \in S$.

Let e and e' be expressions built over Ω and S, where states in S are regarded as unary symbols. We say e' is a *legal transition* from e if there is an Ω-ranked tree t and a transition rule $(\omega, \boldsymbol{s}, s') \in \Theta$ such that $e \neq e'$ and e' is obtained from e by substituting $s'(\omega(\boldsymbol{t}))$ for exactly one occurrence of $\omega(s_1(t_1), \ldots, s_k(t_k))$, where both $\boldsymbol{s}$ and $\boldsymbol{t}$ are k-tuples for some k, and s_i (resp., t_i) is the i-th component of $\boldsymbol{s}$ (resp., $\boldsymbol{t}$). Every *run* of $\mathcal{A}$ on an Ω-ranked tree t is a finite sequence of expressions $e_0, \ldots, e_n$ such that $e_0 = t$, e_i is a legal transition from e_{i-1} for $0 < i \leq n$, and there is no legal transition from e_n.

An NTA $\mathcal{A} = (S, F, \Omega, \Theta)$ is said to *accept* an Ω-ranked tree $\mathfrak{T}$ if there is a run $e_0, \ldots, e_n$ of $\mathcal{A}$ on $\mathfrak{T}$ and a final state $s \in F$ such that $e_n = s(\mathfrak{T})$. An Ω-ranked tree language $\mathbb{L}$, i.e., a set of Ω-ranked trees, is said to be *recognized* by $\mathcal{A}$ if every Ω-ranked tree is accepted by $\mathcal{A}$ if, and only if, it is in $\mathbb{L}$. It is well-known that a ranked tree language is recognized by some NTA iff it is regular, see, e.g., (Comon et al. 2007).

An NTA $\mathcal{A}$ is called *oblivious* if for every pair of transition rules $(\omega, \boldsymbol{s}, s_0)$ and $(\omega', \boldsymbol{s}', s_0')$ of $\mathcal{A}$, if $\ell(\omega) = \ell(\omega')$ then we have $s_0 = s_0'$. In other words, the transition of $\mathcal{A}$ only depends on the letter of the current input symbol. Given a ranked tree $\mathfrak{T} = (V, E, r, L)$, the *accompanying tree* of $\mathfrak{T}$, denoted $\ell(\mathfrak{T})$, is defined as the labeled tree $(V, E, r, \ell(L))$ where $\ell(L)(v) = \ell(L(v))$ for all $v \in V$. Given a ranked tree language $\mathbb{L}$, the *accompanying tree language* of $\mathbb{L}$ is the class of $\ell(\mathfrak{T})$ for all $\mathfrak{T} \in \mathbb{L}$. Interestingly, a ranked tree language is recognized by an oblivious NTA iff it is regular and its accompanying tree language is closed under prefixes.

Automata That Accept BCQs Let q be a BCQ. Let $\mathcal{L}_q$ denote the set of order pairs $\langle X, \Phi\rangle$ where X is a finite set of variables or constants, and $\Phi \subseteq [q]$. A *tree representation* of q is a finite $\mathcal{L}_q$-labeled tree $\mathfrak{R} = (V, E, r, L)$ such that

1. $[q] = \bigcup_{v \in V} L^2(v)$, and $term(L^2(v)) \subseteq L^1(v)$ for every $v \in V$, where $L^i(v)$ denotes the i-th component of $L(v)$;
2. the subgraph of $\mathfrak{R}$ induced by the set $\{v \in V : t \in L^1(v)\}$ is connected for every $t \in \Delta \cup \Delta_{\mathrm{v}}$;
3. for all $v \in V$, all constants in $L^1(v)$ also occur in $L^1(r)$.

The *width* of $\mathfrak{R}$ is the maximum cardinality of $L^1(v)$ for all $v \in V$. In particular, a tree representation $\mathfrak{R} = (V, E, r, L)$ of q is called *linear* if, for each $v \in V$, we have $|L^2(v)| \leq 1$.

Note that a tree representation of q is not necessary a tree decomposition, but based on any tree decomposition of q, one can easily construct a tree representation.

Next we show how to encode BCQs as inputs of an NTA. Let $\mathscr{Q}$ be a schema and q a $\mathscr{Q}$-BCQ. Let $\mathfrak{R} = (V, E, r, L)$ be a tree representation of q. A rough idea of encoding q is by directly regarding $\mathfrak{R}$ as the accompanied tree of a ranked tree. However, this is infeasible because the ranked input alphabet is required to be finite, while the BCQs that we have to consider may involve an unbounded number of terms.

A natural idea to resolve the mentioned issue is by reusing variables. For example, suppose v_1, v_2 and v_3 are nodes in $\mathfrak{R}$ such that v_2 is a child of v_1, and v_3 a child of v_2. Suppose

$$\begin{aligned}
L(v_1) &= (\{x_1, x_2, x_3\}, \{R(x_1, x_2, x_3)\}),\\
L(v_2) &= (\{x_2, x_3, x_4\}, \{S(x_3, x_4)\}),\\
L(v_3) &= (\{x_3, x_4, x_5\}, \{T(x_5, x_4, x_5)\}).
\end{aligned}$$

By the definition of tree representation, x_1 is not allowed to appear in v_3 and its descendants. We thus can reuse x_1 in v_3, and let $L(v_3) = (\{x_3, x_4, x_1\}, \{T(x_1, x_4, x_1)\})$. We assume all the variables occurring in v_3 but not in v_2 are fresh variables. Clearly, by reusing variables, only $2k$ variables are needed to encode a tree representation of the width k.

Let $\mathcal{V}$ be a set that consists of $2k$ variables. Let At denote the set of $\mathscr{Q}$-atoms involving terms only from $const(q)$ and $\mathcal{V}$. Let $\mathcal{L}$ be a label set consisting of all the pairs $\omega = (X, \Phi)$ such that $X \subseteq const(q) \cup \mathcal{V}$ and Φ is either $\emptyset$ or $\{\alpha\}$ for some $\alpha \in At$. Clearly, $\mathcal{L}$ is finite. By the technique of reusing variables, $\mathfrak{R}$ can be represented as an $\mathcal{L}$-labeled tree. Suppose $\mathfrak{R}' = (V, E, r, L')$ is the mentioned tree. Let $\mathfrak{T}$ denote the ranked tree (V, E, r, L^*) where $L^*(v) = (L'(v), n)$ if $v \in V$ has exactly n children. Clearly, from $\mathfrak{T}$ one can easily obtain q. We call $\mathfrak{T}$ a *ranked tree representation* of q.

We say an NTA $\mathcal{A}$ *accepts* q if $\mathcal{A}$ accepts $\mathfrak{T}$ for some ranked tree representation $\mathfrak{T}$ of q, and $\mathcal{A}$ *recognizes* a class $\mathcal{C}$ of $\mathscr{Q}$-BCQs if for all $\mathscr{Q}$-BCQs q, $\mathcal{A}$ accepts q iff $q \in \mathcal{C}$.

Characterization We first define some notions and notations. A BCQ q is said to be *nontrivial* if $[q] \neq \emptyset$, and q is called a *proper subquery* of another BCQ p if $[q] \subsetneq [p]$. A BCQ q is said to be *inseparable* if there are no nontrivial proper subqueries q_1 and q_2 of q such that q is equivalent to $q_1 \wedge q_2$. Let $\mathcal{C}$ be a class of BCQs. A BCQ $q \in \mathcal{C}$ is said to be *most specific* w.r.t. $\mathcal{C}$ if $s(q) \notin \mathcal{C}$ for each partial function $s : \Delta_{\mathrm{v}} \to \Delta$ that maps at least one variable occurring in q to a constant. Furthermore, a BCQ $q \in \mathcal{C}$ is said to be *prime* w.r.t. $\mathcal{C}$ if it is inseparable and most specific w.r.t. $\mathcal{C}$.

Given an OMQA[CQ]-ontology O and a database D, let $O(D)$ denote the class of BCQs q such that $(D, q) \in O$.

Now we have a characterizations for linear TGDs.

Theorem 16. *Let $\mathscr{D}$ and $\mathscr{Q}$ be schemas. An OMQA*[CQ]-*ontology O over $(\mathscr{D}, \mathscr{Q})$ is defined by a finite set of linear TGDs iff it admits both of the following properties:*

1. *(Data Constructivity) If D and D' are $\mathscr{D}$-databases and $q \in O(D \cup D')$ is prime w.r.t. $O(D \cup D')$, then we have either $q \in O(D)$ or $q \in O(D')$.*
2. *(NTA-recognizability) If D is a $\mathscr{D}$-database with a single fact, then $O(D)$ is recognized by some oblivious NTA.*

Sketch of Proof. Due to space limit, we only give a proof for the direction of "only-if". Suppose O is defined by a finite set Σ of linear TGDs. We need to show that O admits Properties 1 and 2. Property 1 can be proven by a careful induction on the chase. Below we prove that O admits Property 2.

Let D be a $\mathscr{D}$-database with a single fact. Now let us construct an oblivious NTA that recognizes $O(D)$. Let $\mathscr{S}$ denote the schema of Σ. Let k be the maximum arity of relation symbols in $\mathscr{S}$. Let $\mathcal{V}$ be the set that consists of pairwise distinct variables $x_1, \dots, x_{2k}$. Let At be the set of all atoms built upon relation symbols from $\mathscr{S}$ and terms from $adom(D) \cup \mathcal{V}$. We introduce $\lceil \log_2(|At|+2) \rceil$ fresh variables, and let $\mathcal{V}_0$ denote the set that consists of these variables. Let ι be an injective function from At to $2^{\mathcal{V}_0} \setminus \{\emptyset, \mathcal{V}_0\}$. Thus, every atom in At can be encoded by a set of variables in $\mathcal{V}_0$.

With the above assumptions, we are now able to define the NTA. Let $S = At \cup \{\diamond\}$ be the set of states, and let $F = \{\diamond\}$ be the set of final state where $\diamond$ is used as the unique final state. Furthermore, let $\mathcal{L}$ be a label set which consists of

1. $(term(\alpha), \{\alpha\})$ for each $\alpha \in At$ which is a $\mathscr{Q}$-atom;
2. $(term(\alpha) \cup \iota(\alpha), \emptyset)$ for each $\alpha \in At$;
3. $(adom(D) \cup \mathcal{V}_0, \emptyset)$.

For convenience, let $\lambda : \mathcal{L} \to S$ be a function that maps each label $\ell \in \mathcal{L}$ of the form 1 or 2 to the atom (state) α, and maps the label of the form 3 to the final state $\diamond$. Clearly, λ is well-defined. Let Ω be a ranked input alphabet which consists of ordered pairs (ℓ, m) for all $\ell \in \mathcal{L}$ and all $0 \leq m \leq |At|$, where each (ℓ, m) is used as an m-ary input symbol.

Furthermore, let Θ be a set consisting of

1. $((\ell, 1), \alpha, \diamond)$ if $\lambda(\ell) = \diamond$ and $D = \{\alpha\}$;
2. $((\ell, m), (\alpha_1, \dots, \alpha_m), \alpha)$ if $\lambda(\ell) = \alpha$, $0 \leq m \leq |At|$, $\alpha, \alpha_1, \dots, \alpha_m \in At$ and for $1 \leq i \leq m$, $\{\alpha\} \cup \Sigma \models \exists \boldsymbol{x}_i \alpha_i$, where $\boldsymbol{x}_i$ is a tuple consisting of all the variables that occur in α_i but not in α.

Let $\mathcal{A} = (S, F, \Omega, \Theta)$. Since λ is a well-defined function, we know that $\mathcal{A}$ is an oblivious NTA. Next we show that $\mathcal{A}$ recognizes the class $O(D)$. By the definition of $\mathcal{A}$, it is easy to see that every $\mathscr{Q}$-BCQ accepted by $\mathcal{A}$ belongs to $O(D)$.

Conversely, let $q \in O(D)$. We need to prove that $\mathcal{A}$ accepts q. Let $\mathfrak{D}$ be a labeled tree constructed as follows:

1. Create the root r with the label $L(r) = (adom(D), D)$;
2. For each node v already in $\mathfrak{D}$, if there is an atom $\alpha \in At$ such that $L^2(v) \cup \Sigma \models \exists \boldsymbol{x}\, \alpha$, then create a child v' for v and let $L(v') = (term(\alpha^*), \{\alpha^*\})$, where α^* is obtained from α by substituting fresh variables for variables in $\boldsymbol{x}$.

Let $atom(\mathfrak{D})$ be the set of all atoms appearing in $\mathfrak{D}$. By definition we know that $chase(D, \Sigma)$ is $adom(D)$-isomorphic to a subset of $atom(\mathfrak{D})$. Let C denote $const(q)$. As $q \in O(D)$, according to the construction of $\mathfrak{D}$, it is not difficult to prove that $[q]$ is C-isomorphic to a subset of $atom(\mathfrak{D})$.

Let Q be a subset of $atom(\mathfrak{D})$ that is C-isomorphic to $[q]$. Let $\mathfrak{D}_q$ be a minimal connected subgraph of $\mathfrak{D}$ that covers Q and the root r. Suppose $\mathfrak{D}_q = (V, E, r, L)$. Next, let $\mathfrak{R}_q$ be the labeled tree (V, E, r, L_0) where L_0 is defined as follows:

1. for the root r, let $L_0(r) = (adom(D) \cup \mathcal{V}_0, \emptyset)$;
2. for every node $v \in V \setminus \{r\}$ with $L(v) = (term(\alpha), \{\alpha\})$, let $L_0(v) = (term(\alpha) \cup \iota(\alpha), \emptyset)$ if $\alpha \notin Q$, and $L_0(v) = L(v)$ otherwise.

Clearly, $\mathfrak{R}_q$ is a finite and linear tree representation of q. By the technique mentioned in the last subsection, such a tree can be naturally encoded by an Ω-ranked tree, which can be easily showed to be accepted by $\mathcal{A}$ by a routine check. $\square$

Conclusion and Related Work

We have established a number of novel characterizations for the program expressive power of DTGDs, TGDs as well as linear TGDs. These results make significant contributions towards a complete picture for the (absolute) program expressive power of existential rule languages. As a byproduct, we have proposed a new chase procedure called nondeterministic chase for DTGDs. Moreover, we have observed that queries derivable from linear TGDs are recognizable by a natural class of tree automata, and this may shed light on optimizing ontology by automata techniques.

Besides the data and program expressive power, there has been some earlier research motivated to characterize other kinds of expressive power of existential rule languages. For example, ten Cate and Kolaitis (2010) characterized the source-to-target TGDs (a class of acyclic TGDs) and its subclasses under the semantics of schema mapping; by regarding ontology languages as logical languages, (Makowsky and Vardi 1986; Zhang, Zhang, and Jiang 2020; Console, Kolaitis, and Pieris 2021) established a number of model-theoretic characterizations for existential rule languages, including DEDs, DTGDs, TGDs, equality-generating dependencies, full TGDs, guarded TGDs as well as linear TGDs.

Acknowledgements

We would like to thank anonymous referees for their helpful comments and suggestions. This work was partially supported by the National Key Research and Development Program of China (2020AAA0108504) and the National Natural Science Foundation of China (61806102, 61972455).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Arenas, M.; Gottlob, G.; and Pieris, A. 2014. Expressive languages for querying the semantic web. In *Proceedings of PODS-2014*, 14–26.

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.

Beeri, C.; and Vardi, M. Y. 1981. The Implication Problem for Data Dependencies. In *Proceedings of ICALP-1981*, 73–85.

Bellomarini, L.; Gottlob, G.; Pieris, A.; and Sallinger, E. 2017. Swift Logic for Big Data and Knowledge Graphs. In *Proceedings of IJCAI-2017*, 2–10.

Bienvenu, M.; ten Cate, B.; Lutz, C.; and Wolter, F. 2014. Ontology-Based Data Access: A Study through Disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4): 33:1–33:44.

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.; Lukasiewicz, T.; Marnette, B.; and Pieris, A. 2010. Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. In *Proceedings of LICS-2010*, 228–242.

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

Comon, H.; Dauchet, M.; Gilleron, R.; Löding, C.; Jacquemard, F.; Lugiez, D.; Tison, S.; and Tommasi, M. 2007. *Tree Automata Techniques and Applications*. Online Book.

Console, M.; Kolaitis, P. G.; and Pieris, A. 2021. Model-theoretic Characterizations of Rule-based Ontologies. In *Proceedings of PODS-2021*, 416–428.

Fagin, R.; Kolaitis, P.; Miller, R. J.; and Popa, L. 2005. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1): 89–124.

Gottlob, G.; Rudolph, S.; and Simkus, M. 2014. Expressiveness of guarded existential rule languages. In *Proceedings of PODS-2014*, 27–38.

Krötzsch, M.; and Rudolph, S. 2011. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In *Proceedings of IJCAI-2011*, 963–968.

Lenzerini, M. 2002. Data Integration: A Theoretical Perspective. In *Proceedings of PODS-2002*, 233–246.

Leone, N.; Manna, M.; Terracina, G.; and Veltri, P. 2012. Efficiently Computable Datalog$^\exists$ Programs. In *Proceedings of KR-2012*.

Makowsky, J. A.; and Vardi, M. Y. 1986. On the Expressive Power of Data Dependencies. *Acta Inf.*, 23(3): 231–244.

Marnette, B. 2009. Generalized schema-mappings: from termination to tractability. In *Proceedings of PODS-2009*, 13–22.

Rudolph, S.; and Thomazo, M. 2015. Characterization of the Expressivity of Existential Rule Queries. In *Proceedings of IJCAI-2015*, 3193–3199.

ten Cate, B.; and Kolaitis, P. 2010. Structural characterizations of schema-mapping languages. *Commun. ACM*, 53(1): 101–110.

Zhang, H.; and Zhang, Y. 2017. Expressiveness of Logic Programs under the General Stable Model Semantics. *ACM Trans. Comput. Log.*, 18(2): 9:1–9:28.

Zhang, H.; Zhang, Y.; and Jiang, G. 2020. Model-theoretic Characterizations of Existential Rule Languages. In *Proceedings of IJCAI-2020*, 1940–1946.

Zhang, H.; Zhang, Y.; and You, J. 2015. Existential Rule Languages with Finite Chase: Complexity and Expressiveness. In *Proceedings of AAAI-2015*, 1678–1685.

Zhang, H.; Zhang, Y.; and You, J. 2016. Expressive Completeness of Existential Rule Languages for Ontology-Based Query Answering. In *Proceedings of IJCAI-2016*, 1330–1337.

Zhang, H.; Zhang, Y.; You, J.; Feng, Z.; and Jiang, G. 2020. Towards Universal Languages for Tractable Ontology Mediated Query Answering. In *Proceedings of AAAI-2020*, 3049–3056.