

The Impact of Active Domain Predicates on Guarded Existential Rules

Georg Gottlob¹, Andreas Pieris², and Mantas Šimkus²

¹ Department of Computer Science, University of Oxford georg.gottlob@cs.ox.ac.uk

² Institute of Information Systems, TU Wien {pieris,simkus}@dbai.tuwien.ac.at

Abstract. We claim it is realistic to assume that a database management system provides access to the active domain via built-in relations. Therefore, product databases, i.e., databases that include designated predicates that hold the active domain, form a natural notion that deserves our attention. An important issue then is to look at the consequences of product databases for the expressiveness and complexity of central existential rule languages. We focus on guarded existential rules, and we investigate the impact of product databases on their expressive power and complexity. We show that the queries expressed via (frontier-)guarded rules gain in expressiveness, and in fact, they have the same expressive power as Datalog. On the other hand, there is no impact on the expressiveness of the queries specified via weakly-(frontier-)guarded rules since they are powerful enough to explicitly compute the predicates needed to access the active domain. We also observe that there is no impact on the complexity of the languages in question.

1 Introduction

Rule-based languages lie at the core of databases and knowledge representation. In database applications they are usually employed as query languages that go beyond standard SQL, while in knowledge representation are used for declarative problem solving, and, more recently, to model and reason about ontological knowledge. Therefore, rule-based languages can be used in at least two different ways: as query languages and as ontology languages. In the database setting, a rule-based query is expressed as a pair of the form (Σ, Ans) , where Σ is a set of rules encoding the actual query, and Ans is the so-called goal predicate that collects the answer to the query. On the other hand, in the ontological setting, a database D and a set of rules Σ are used to specify implicit domain knowledge – the pair (D, Σ) is called *knowledge base* – while user queries, typically expressed as standard conjunctive queries, are evaluated over a knowledge base. Alternatively, the set of rules can be conceived as part of the specification of a query that is executed over a plain database. Such queries are known as *ontology-mediated queries* [6], and are in fact pairs of the form (Σ, q) , where Σ is a set of rules expressed in a certain ontology language, and q is a conjunctive query. From the above discussion, it is apparent that rule-based languages form the building block of several database and ontology-mediated query languages that can be found in the literature.

An important issue for a query language (either a database or an ontology-mediated query language) is to understand its expressive power, and in particular, its expressiveness relative to other query languages. *Relative expressiveness* considers if, given two

query languages L_1 and L_2 , every query formulated in L_1 can be expressed by means of L_2 (and vice versa). This helps the user to choose, among a plethora of different query languages, the one that is more appropriate for the application in question. The goal of this work is to perform such an expressivity analysis for central query languages based on existential rules.

Existential rules (a.k.a. *tuple-generating dependencies* or *Datalog[±] rules*) are first-order sentences of the form $\forall \bar{x} \forall \bar{y} (\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \psi(\bar{x}, \bar{z}))$, where ϕ and ψ are conjunctions of atoms. Intuitively speaking, such a rule states that the existence of certain tuples in a database implies the existence of some other tuples in the same database. It is widely known that the query languages based on arbitrary existential rules, without posing any syntactic restriction, are undecidable; see, e.g., [5, 7]. This has led to a flurry of activity for identifying expressive fragments of existential rules that give rise to decidable query languages. One of the key paradigms that has been thoroughly studied is guardedness [2, 7]. In a nutshell, the existential rule given above is guarded (resp., frontier-guarded) if ϕ has an atom that contains (or “guards”) all the variables in $\bar{x} \cup \bar{y}$ (resp., \bar{x}). More refined languages based on weak-(frontier-)guardedness also exist.

The relative expressiveness of the languages based on (weakly-)(frontier-)guarded existential rules has been recently investigated in [9]. However, the thorough analysis performed in [9] has made no assumption on the input databases over which the queries will be evaluated, and it is known that such assumptions may have an impact on the expressiveness of a query language. Recall the classical result that semipositive Datalog over ordered databases is powerful enough to express all queries that are computable in polynomial time, which is not true without assuming ordered databases [1].

We claim it is natural to focus on *product databases*, that is, databases that include designated predicates that hold the active domain. In other words, those predicates give access to the cartesian product of the active domain (hence the name product databases). Since it is realistic to assume that a database management system provides access to the active domain via built-in relations (e.g., lookup or reference tables), we believe that product databases form a central notion that deserves our attention. In view of this fact, it is important to understand how the relative expressiveness of the guarded-based query languages in question is affected when we concentrate on product databases. This is the goal of the present work. The outcome of our analysis can be summarized as follows:

- The query languages based on (frontier-)guarded existential rules gain in expressiveness, and, in fact, they have the same expressive power as Datalog.
- There is no impact on the expressive power of the query languages that are based on weakly-(frontier-)guarded existential rules, since they are powerful enough to explicitly compute the relations needed to access the active domain.
- Finally, we show that there is no impact on the computational complexity of the guarded-based query languages in question.

Although the employed techniques for establishing the above results are rather standard, which build on existing ones that can be found in the literature, the obtained results are conceptually interesting (e.g., assuming product databases, (frontier-)guardedness gives rise to query languages that are equally expressive to Datalog). We believe that our analysis sheds light on the expressivity of the guarded-based query languages in question, and complements the recent investigation performed in [9]. Let us clarify that in the

above summarization of our results, the term query language refers to both database and ontology-mediated query languages. Since the former is a special case of the latter (indeed, the query (Σ, Ans) is actually the ontology-mediated query $(\Sigma, \text{Ans}(x_1, \dots, x_n))$, where n is the arity of Ans), in the sequel we focus on ontology-mediated queries.

2 Motivating Example

The goal of this section is to illustrate, via a meaningful example, that product databases have an impact on the expressiveness of frontier-guarded ontology-mediated queries, which in turn allows us to write complex queries in a more flexible way. Suppose we are developing a system for managing a response to a natural disaster. The ultimate goal of the system is to collect information about volunteers and their qualifications, and then use this information to coordinate various relief activities.

The Database. Suppose that the database of such a system contains a binary relation `Team` that stores an assignment of volunteers to teams. For example, the atom

`Team("Alpha", "Ann")`

means that Ann belongs to the team called Alpha. The database also includes a binary relation called `ExperienceIn`, which relates persons to tasks in which they have experience. For instance, the atom

`ExperienceIn("John", "perform CPR")`

states that John has experience in performing CPR. We also have a binary relation `hasTraining` with the obvious meaning; for example,

`hasTraining("John", "race driver")`

means that John has been trained to drive a race car. In addition, the database contains a unary relation `ProDriverQualification` that stores qualifications that involve driving at professional level; e.g.,

`ProDriverQualification("bus license")`

states that bus license is a qualification to drive at professional level. We further assume that some tasks that can be performed by volunteers are grouped into more complex procedures. For instance, the response to a water leak could consist of performing four tasks in the following order: load equipment, drive truck, perform repairs and clean up. This is stated in the database of the system using the atoms:

`ProcedureTaskFirst("water leak", "load equipment")`
`ProcedureTaskOrder("water leak", "load equipment", "drive truck")`
`ProcedureTaskOrder("water leak", "drive truck", "perform repairs")`
`ProcedureTaskOrder("water leak", "perform repairs", "clean up")`
`ProcedureTaskLast("water leak", "clean up").`

Intuitively, $\text{ProcedureTaskFirst}(p, t)$ and $\text{ProcedureTaskLast}(p, t')$ state that t/t' are the first/last task in the procedure p . The atom $\text{ProcedureTaskOrder}(p, t, t')$ says that in the procedure p the task t' follows the task t .

The Ontology. We know that some intensional knowledge, not explicitly stored in the database described above, also holds. More precisely, we know that if a person p has experience in some task t , then p is qualified to perform t . This can be expressed as

$$\sigma_1 = \text{ExperienceIn}(Pn, Tk) \rightarrow \text{QualifiedFor}(Pn, Tk).$$

Moreover, we know that if a person p has been trained to be a professional driver, then p is qualified to drive an ambulance. This can be expressed as

$$\sigma_2 = \text{hasTraining}(Pn, T), \text{ProDriverQualification}(T) \rightarrow \text{QualifiedFor}(Pn, \text{"drive ambulance"}).$$

In addition, if a person p is experienced in delivering heavy goods, then p must have some training that leads to a truck license. This is expressed via the rule

$$\sigma_3 = \text{ExperienceIn}(Pn, \text{"delivery heavy goods"}) \rightarrow \exists T \text{hasTraining}(Pn, T), \text{TruckLicense}(T).$$

Finally, truck license leads to a professional driving license, which can be expressed as

$$\sigma_4 = \text{TruckLicense}(T) \rightarrow \text{ProDriverQualification}(T).$$

Observe that our ontology $\Sigma = \{\sigma_1, \dots, \sigma_4\}$ consists of guarded existential rules.

The Database Query. In our disaster management scenario we are interested in checking whether a team is qualified to perform every task of a certain procedure. More precisely, we want to collect in a binary relation TeamQualified all pairs (t, p) of a team and a procedure such that: for every task j of the procedure p , the team t has a member m that is qualified for j . Recall that an ontology-mediated query is a pair of an ontology and a database query. Therefore, we need to express the above query as a database query q , which, together with the ontology Σ defined above, will give rise to the ontology-mediated query (Σ, q) . Unfortunately, things are a bit more complicated than they seem. In particular, the query q is inherently recursive, and thus is not expressible as a conjunctive query. However, it can be easily expressed as the Datalog query $(\Pi, \text{TeamQualified})$, where the program Π consists of the rules:

$$\begin{aligned} & \text{ProcedureTaskFirst}(Pc, Tk), \\ \rho_1 = & \text{Team}(Tm, Pn), \\ & \text{QualifiedFor}(Pn, Tk) \rightarrow \text{QualifiedUntil}(Tm, Pc, Tk) \\ \\ & \text{ProcedureTaskOrder}(Pc, Tk', Tk), \\ \rho_2 = & \text{Team}(Tm, Pn), \\ & \text{QualifiedUntil}(Tm, Pn, Tk') \rightarrow \text{QualifiedUntil}(Tm, Pc, Tk) \\ \\ \rho_3 = & \text{ProcedureTaskLast}(Pc, Tk), \\ & \text{QualifiedUntil}(Tm, Pc, Tk) \rightarrow \text{TeamQualified}(Tm, Pc). \end{aligned}$$

The fact that our query q is expressible as a recursive Datalog query is of little use since the ontology-mediated query (Σ, q) does not comply with the formal definition of ontology-mediated queries where q must be a first-order query, and thus does not fall in a decidable guarded-based ontology-mediated query language. Hence, the crucial question that comes up is whether we can construct a query (Σ', q') that is equivalent to (Σ, q) , while Σ' is a set of (frontier-)guarded existential rules and q is a conjunctive query. One may think that this can be achieved by adding the rules of Π in the ontology Σ , i.e., $\Sigma' = \Sigma \cup \Pi$, and let q be the atomic conjunctive query $\text{TeamQualified}(x, y)$. Although the obtained query (Σ', q') is equivalent to (Σ, q) , it is inherently unguarded, and it cannot be expressed as a frontier-guarded ontology-mediated query. However, assuming that our database is product, which gives us access to the active domain via relations of the form Dom^k , for $k > 0$, that hold all the k -tuples of constants occurring in the active domain, we can convert the rules $\rho_1, \rho_2 \in \Sigma'$ into guarded rules, without changing the meaning of the query (Σ', q') , by adding in their body the atom $\text{Dom}^4(Tm, Pc, Tk, Pn)$ and $\text{Dom}^5(Tm, Pc, Tk, Tk', Pn)$, respectively. Hence, the assumption that the database is product allows us to rewrite the query (Σ, q) into an equivalent guarded ontology-mediated query.

3 Preliminaries

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

A *query* over \mathbf{S} is a mapping q that maps every database D over \mathbf{S} to a set of *answers* $q(D) \subseteq \text{adom}(D)^n$, where $n \geq 0$ is the *arity* of q . The usual way of specifying queries is by means of (fragments of) first-order logic. Such a central fragment is the class of conjunctive queries. A *conjunctive query* (CQ) q over \mathbf{S} is a conjunction of atoms of the form $\exists \bar{y} \phi(\bar{x}, \bar{y})$, where $\bar{x} \cup \bar{y}$ are variables of \mathbf{V} , that uses only predicates from \mathbf{S} . The free variables of a CQ are called *answer variables*. The evaluation of CQs over instances is defined in terms of homomorphisms. A *homomorphism* from a set of atoms A to a set of atoms A' is a partial function $h : \mathbf{C} \cup \mathbf{N} \cup \mathbf{V} \rightarrow \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ such that: (i) $t \in \mathbf{C}$ implies $h(t) = t$, i.e., is the identity on \mathbf{C} , and (ii) $R(t_1, \dots, t_n) \in A$ implies $h(R(t_1, \dots, t_n)) = R(h(t_1), \dots, h(t_n)) \in A'$. The *evaluation* of q over an \mathbf{S} -instance I , denoted $q(I)$, is the set of all tuples $h(\bar{x})$ of constants such that h is a homomorphism from q to I . Each schema \mathbf{S} and CQ $q = \exists \bar{y} \phi(x_1, \dots, x_n, \bar{y})$ give rise to the n -ary query $q_{\phi, \mathbf{S}}$ defined by setting, for every database D over \mathbf{S} , $q_{\phi, \mathbf{S}}(D) = \{\bar{c} \in \text{adom}(D)^n \mid \bar{c} \in q(D)\}$. Let CQ be the class of all queries definable by some CQ.

Tgds for Specifying Ontologies. An ontology language is a fragment of first-order logic. We focus on ontology languages that are based on tuple-generating dependencies.

A *tuple-generating dependency* (tgd) is a first-order sentence of the form

$$\forall \bar{x} \forall \bar{y} (\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \psi(\bar{x}, \bar{z})),$$

where both ϕ and ψ are conjunctions of atoms without nulls and constants. For simplicity, we write this tgd as $\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \psi(\bar{x}, \bar{z})$, and use comma instead of “ \wedge ” for conjoining atoms. We call ϕ and ψ the *body* and *head* of the tgd, respectively, and write $sch(\Sigma)$ for the set of predicates occurring in Σ . An instance I *satisfies* the above tgd if: For every homomorphism h from $\phi(\bar{x}, \bar{y})$ to I , there is a homomorphism h' that extends h , i.e., $h' \supseteq h$, from $\psi(\bar{x}, \bar{z})$ to I . I satisfies a set Σ of tgds, denoted $I \models \Sigma$, if I satisfies every tgd in Σ . Let TGD be the class of all (finite) sets of tgds.

Ontology-Mediated Queries. An *ontology-mediated query* is a triple (\mathbf{S}, Σ, q) , where \mathbf{S} is a schema, called *data schema*, $\Sigma \in \text{TGD}$, $q \in \text{CQ}$, and q is over $\mathbf{S} \cup sch(\Sigma)$.³ Notice that the data schema \mathbf{S} is included in the specification of an ontology-mediated query in order to make clear that the query is over \mathbf{S} , i.e., it ranges over \mathbf{S} -databases. The semantics of such a query is defined in terms of certain answers. Let (\mathbf{S}, Σ, q) be an ontology-mediated query, where n is the arity of q . The *answer* to q with respect to a database D over \mathbf{S} and Σ is $cert_{q, \Sigma}(D) = \bigcap_{I \supseteq D, I \models \Sigma} \{\bar{c} \in adom(D)^n \mid \bar{c} \in q(I)\}$.

At this point, it is important to recall that $cert_{q, \Sigma}(D)$ coincides with the evaluation of q over the canonical instance of D and Σ that can be constructed by applying the chase procedure [7, 8, 10, 11]. Roughly speaking, the chase adds new atoms to D as dictated by Σ until the final result satisfies Σ , while the existentially quantified variables are satisfied by inventing fresh null values. The formal definition of the chase procedure follows. Let I be an instance and $\sigma = \phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \psi(\bar{x}, \bar{z})$ a tgd. We say that σ is *applicable* with respect to I if there exists a homomorphism h from $body(\sigma)$ to I . In this case, *the result of applying σ over I with h* is the instance $J = I \cup h'(head(\sigma))$, where h' is an extension of h that maps each $z \in \bar{z}$ to a fresh null value not in I . For such a single chase step we write $I \xrightarrow{\sigma, h} J$. Let us assume now that I is an instance and Σ a finite set of tgds. A *chase sequence for I under Σ* is a (finite or infinite) sequence: $I_0 \xrightarrow{\sigma_0, h_0} I_1 \xrightarrow{\sigma_1, h_1} I_2 \dots$ of chase steps such that: (1) $I_0 = I$; (2) For each $i \geq 0$, $\sigma_i \in \Sigma$; and (3) $\bigcup_{i \geq 0} I_i \models \Sigma$. Notice that in case the above chase sequence is infinite, then it must be also *fair*, that is, whenever a tgd $\sigma \in \Sigma$ is applicable with respect to I_i with homomorphism h_i , then there exists $h' \supseteq h_i$ and $k > i$ such that $h'(head(\sigma)) \subseteq I_k$. In other words, a fair chase sequence guarantees that all tgds that are applicable will eventually be applied. We call $\bigcup_{i \geq 0} I_i$ the *result* of this chase sequence, which always exists. Although the result of a chase sequence is not necessarily unique (up to isomorphism), each such result is equally useful for our purposes since is *universal*, that is, it can be homomorphically embedded into every other result. Therefore, we denote by $chase(I, \Sigma)$ the result of an arbitrary chase sequence for I under Σ .

Given an ontology-mediated query (\mathbf{S}, Σ, q) , it is well-known that $cert_{q, \Sigma}(D) = q(chase(D, \Sigma))$, for every \mathbf{S} -database D . In other words, to compute the answer to q with respect to D and Σ , we simply need to evaluate q over the instance $chase(D, \Sigma)$. Notice that this does not provide an effective algorithm for computing $cert_{q, \Sigma}(D)$ since the instance $chase(D, \Sigma)$ is, in general, infinite.

³ In fact, ontology-mediated queries can be defined for arbitrary ontology and query languages.

Ontology-Mediated Query Languages. Every ontology-mediated query $Q = (\mathbf{S}, \Sigma, q)$ can be interpreted as a query q_Q over \mathbf{S} by setting $q_Q(D) = \text{cert}_{q, \Sigma}(D)$, for every \mathbf{S} -database D . Thus, we obtain a new query language, denoted (TGD, CQ) , defined as the class of queries q_Q , where Q is an ontology-mediated query. However, (TGD, CQ) is undecidable since, given a database D over \mathbf{S} , $\Sigma \in \text{TGD}$, an n -ary query $q \in \text{CQ}$ over $\mathbf{S} \cup \text{sch}(\Sigma)$, and a tuple $\bar{c} \in \mathbf{C}^n$, the problem of deciding whether $\bar{c} \in \text{cert}_{q, \Sigma}(D)$ is undecidable; see, e.g., [5, 7]. This has led to a flurry of activity for identifying decidable syntactic restrictions. Such a restriction defines a subclass \mathcal{C} of tgds, i.e., $\mathcal{C} \subseteq \text{TGD}$, which in turn gives rise to the query language (\mathcal{C}, CQ) . Such a query language is called *ontology-mediated query language*. Here we focus on ontology-mediated query languages that are based on the notion of guardedness:

(Frontier-)Guarded Tgds: A tgd is *guarded* if its body contains an atom, called *guard*, that contains all the body-variables [7]. Let \mathbf{G} be the class of all finite sets of guarded tgds. A key extension of guarded tgds is the class of *frontier-guarded* tgds, where the guard contains only the frontier variables, i.e., the body-variables that appear in the head [2]. Let \mathbf{FG} be the class of all finite sets of frontier-guarded tgds.

Weak Versions: Both \mathbf{G} and \mathbf{FG} have a weak version: Weakly-guarded [7] and weakly-frontier-guarded [2], respectively. These are highly expressive classes of tgds obtained by relaxing the underlying condition so that only those variables that may unify with null values during the chase are taken into account. In order to formalize these classes of tgds we need some additional terminology. A *position* $R[i]$ identifies the i -th attribute of a predicate R . Given a schema \mathbf{S} , the set of positions of \mathbf{S} is the set $\{R[i] \mid R/n \in \mathbf{S} \text{ and } i \in \{1, \dots, n\}\}$. Given a set Σ of tgds, the set of *affected positions* of $\text{sch}(\Sigma)$, denoted $\text{affected}(\Sigma)$, is inductively defined as follows: (1) If there exists $\sigma \in \Sigma$ such that at position π an existentially quantified variable occurs, then $\pi \in \text{affected}(\Sigma)$; and (2) If there exists $\sigma \in \Sigma$ and a variable V in $\text{body}(\sigma)$ only at positions of $\text{affected}(\Sigma)$, and V appears in $\text{head}(\sigma)$ at position π , then $\pi \in \text{affected}(\Sigma)$. A tgd σ is *weakly-guarded with respect to* Σ if its body contains an atom, called *weak-guard*, that contains all the body-variables that appear only at positions of $\text{affected}(\Sigma)$. The set Σ is *weakly-guarded* if each $\sigma \in \Sigma$ is weakly-guarded with respect to Σ . The class of weakly-frontier-guarded sets of tgds is defined analogously, but considering only the body-variables that appear also in the head of a tgd. We write \mathbf{WG} (resp., \mathbf{WFG}) for the class of all finite weakly-guarded (resp., weakly-frontier-guarded) sets of tgds.

4 Product Databases

Recall that product databases provide access to the active domain via designated built-in predicates. Before proceeding to the next section, where we look at the impact of product databases on the expressive power of the ontology-mediated query languages in question, let us make the notion of a product database more precise.

A database D is said to be α -*product*, where α is a finite set of positive integers, if it includes a designated predicate Dom^i/i , for each $i \in \alpha$, that holds all the i -tuples of constants in $\text{adom}(D)$, or, in other words, the restriction of D over the predicate Dom^i is precisely the set of facts $\{\text{Dom}^i(\bar{t}) \mid \bar{t} \in \text{adom}(D)^i\}$. Given a non-product database

D , we denote by D^α the α -product database $D \cup \{\text{Dom}^i(\bar{t}) \mid \bar{t} \in \text{adom}(D)^i\}_{i \in \alpha}$. An *ontology-mediated query over a product database* is an ontology-mediated query (\mathbf{S}, Σ, q) such that \mathbf{S} contains the predicates $\text{Dom}^{i_1}, \dots, \text{Dom}^{i_k}$, for some set of positive integers $\alpha = \{i_1, \dots, i_k\}$, while none of those predicates appears in the head of a tgds of Σ . The latter condition is posed since the predicates $\text{Dom}^{i_1}, \dots, \text{Dom}^{i_k}$ are conceived as built-in read-only predicates, and thus, we cannot modify their content. Such a query ranges only over \mathbf{S} -databases that are α -product. We write $(\mathcal{C}, \text{CQ})_\times$ for the class of (\mathcal{C}, CQ) queries over a product database.

Example 1. Consider the query $Q_{\text{trans}} = (\{E\}, \Sigma, \text{Ans}(x, y))$, where Σ is the set:

$$\begin{aligned} E(x, y) &\rightarrow T(x, y) \\ E(x, y), T(y, z) &\rightarrow T(x, z) \\ T(x, y) &\rightarrow \text{Ans}(x, y), \end{aligned}$$

which computes the transitive closure of the binary predicate E . It is easy to see that the above query can be equivalently rewritten as a guarded ontology-mediated query over a product database, i.e., as a $(\mathbf{G}, \text{CQ})_\times$ query. More precisely, Q_{trans} can be written as $Q'_{\text{trans}} = (\{E, \text{Dom}^3\}, \Sigma', \text{Ans}(x, y))$, where Σ' is the set of tgds:

$$\begin{aligned} E(x, y) &\rightarrow T(x, y) \\ \text{Dom}^3(x, y, z), E(x, y), T(y, z) &\rightarrow T(x, z) \\ T(x, y) &\rightarrow \text{Ans}(x, y). \end{aligned}$$

Clearly, for every $\{E\}$ -database D , $Q_{\text{trans}}(D) = Q'_{\text{trans}}(D^{\{3\}})$. ■

5 The Impact of Product Databases

We are now ready to investigate the impact of product databases on the relative expressiveness of the guarded-based ontology-mediated query languages in question. Let us first fix some auxiliary terminology. Two ontology-mediated queries $Q_1 = (\mathbf{S}_1, \Sigma_1, q_1)$ and $Q_2 = (\mathbf{S}_2, \Sigma_2, q_2)$ over a product database, with $\alpha_i = \{j \mid \text{Dom}^j \in \mathbf{S}_i\}$, for each $i \in \{1, 2\}$, are *comparable relative to schema \mathbf{S}* if $\mathbf{S} = \mathbf{S}_1 \setminus \{\text{Dom}^i \mid i \in \alpha_1\} = \mathbf{S}_2 \setminus \{\text{Dom}^i \mid i \in \alpha_2\}$. Such comparable queries are *equivalent*, written $Q_1 \equiv Q_2$, if, for every database D over \mathbf{S} , $\text{cert}_{q_1, \Sigma_1}(D^{\alpha_1}) = \text{cert}_{q_2, \Sigma_2}(D^{\alpha_2})$. It is important to say that the above definitions immediately apply even if we consider queries that are not over a product database. An ontology-mediated query language \mathcal{Q}_2 is *at least as expressive as* the ontology-mediated query language \mathcal{Q}_1 , written $\mathcal{Q}_1 \preceq \mathcal{Q}_2$, if, for every $Q_1 \in \mathcal{Q}_1$ there is $Q_2 \in \mathcal{Q}_2$ such that Q_1 and Q_2 are comparable (relative to some schema) and $Q_1 \equiv Q_2$. \mathcal{Q}_2 is *strictly more expressive than* \mathcal{Q}_1 , written $\mathcal{Q}_1 \prec \mathcal{Q}_2$, if $\mathcal{Q}_1 \preceq \mathcal{Q}_2 \not\preceq \mathcal{Q}_1$. \mathcal{Q}_1 and \mathcal{Q}_2 *have the same expressive power*, written $\mathcal{Q}_1 = \mathcal{Q}_2$, if $\mathcal{Q}_1 \preceq \mathcal{Q}_2 \preceq \mathcal{Q}_1$. In our analysis we also include Datalog. Recall that a Datalog program is simply a set of single-head tgds without existentially quantified variables, while a Datalog query over \mathbf{S} of the form $(\Sigma, \text{Ans}/n)$, where Σ is a Datalog program and Ans is the answer predicate, can be seen as the ontology-mediated query $(\mathbf{S}, \Sigma, \text{Ans}(x_1, \dots, x_n))$. We write DAT for the class of queries definable via some Datalog query. We show the following:

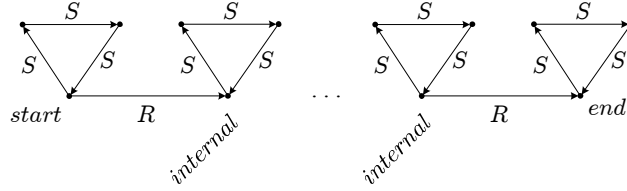


Fig. 1. The graph from the proof of Lemma 1.

Theorem 1. *It holds that,*

$$(G, CQ) \prec (FG, CQ) \prec (G, CQ)_{\times} = (FG, CQ)_{\times} = DAT \prec (WG, CQ) = (WFG, CQ) = (WG, CQ)_{\times} = (WFG, CQ)_{\times}.$$

The rest of this section is devoted to establish the above result. This is done by establishing a series of technical lemmas that all together imply Theorem 1. Henceforth, we assume that, given an ontology-mediated query (S, Σ, q) , none of the predicates of S occur in the head of a tg d of Σ . This assumption can be made without loss of generality since, for each $R \in S$ that appears in the head of tg d of Σ , we can add to Σ the auxiliary copy tg d $R(x_1, \dots, x_n) \rightarrow R^*(x_1, \dots, x_n)$, and then replace each occurrence of R in Σ and q with R^* . We first establish that frontier-guarded ontology-mediated queries are strictly more expressive than guarded ontology-mediated queries. Although this is generally known, it is not explicitly shown in some previous work. Hence, for the sake of completeness, we would like to provide a proof sketch for this fact.

Lemma 1. $(G, CQ) \prec (FG, CQ)$.

Proof (sketch). We need to exhibit a query that can be expressed in (FG, CQ) but not in (G, CQ) , which in turn shows that $(FG, CQ) \not\prec (G, CQ)$; the other direction holds trivially since $G \subseteq FG$. Such a query is the one that asks whether a labeled directed graph $G = (N, E, \lambda, \mu)$, where $\lambda : N \rightarrow \{start, internal, end\}$ and $\mu : E \rightarrow \{R, S\}$, contains a directed R -path P from a start node to an end node via internal nodes, while each node of P is part of a directed S -triangle. In other words, we ask if the graph G contains a subgraph as the one depicted in Figure 1. The graph G is naturally encoded in an S -database D , where $S = \{Start/1, Internal/1, End/1, R/2, S/2\}$. Our query can be expressed as the (FG, CQ) query $Q = (S, \Sigma, \text{Yes}())$, where Σ consists of:

$$\begin{aligned} S(x, x_1), S(x_1, x_2), S(x_2, x) &\rightarrow \text{Triangle}_S(x) \\ \text{Start}(x) &\rightarrow \text{Mark}(x) \\ \text{Mark}(x), \text{Triangle}_S(x), R(x, y), \text{Internal}(y) &\rightarrow \text{Mark}(y) \\ \text{Mark}(x), \text{Triangle}_S(x), R(x, y), \text{End}(y), \text{Triangle}_S(y) &\rightarrow \text{Yes}(). \end{aligned}$$

Let us intuitively explain why Q cannot be expressed as a (G, CQ) query. Assume that Q can be expressed via the (G, CQ) query (S, Σ', q') . It is well-known that the query that asks whether a node belongs to an S -triangle is unguarded, and thus, it cannot be expressed via a query of the form (S, Σ', q_a) , where $\Sigma' \in G$ and q_a is atomic. Thus, the

“triangle checks” must necessarily be performed by the CQ q' . This implies that q' can perform an unbounded number of “triangle checks”, and thus, q' can express a query that is inherently recursive. But this contradicts the fact that a (finite) first-order query, let alone a conjunctive query, cannot express a recursive query. \square

We proceed to show that Datalog queries are strictly more expressive than frontier-guarded ontology-mediated queries. Towards this end, we are going to exploit the fact that $(\text{FG}, \text{ACQ}) \preceq \text{DAT}$, where ACQ is the class of queries definable by some atomic CQ [9].⁴ This means that, given a query $Q = (\mathbf{S}, \Sigma, \exists \bar{y} \text{Ans}(\bar{x}, \bar{y})) \in (\text{FG}, \text{ACQ})$, there exists a procedure Ξ that translates Σ into a Datalog program such that Q and the query $(\Xi(\Sigma), \text{Ans}) \in \text{DAT}$ over \mathbf{S} are equivalent.

Lemma 2. $(\text{FG}, \text{CQ}) \prec \text{DAT}$.

Proof (sketch). We first show that $(\text{FG}, \text{CQ}) \preceq \text{DAT}$. Let $Q = (\mathbf{S}, \Sigma, q) \in (\text{FG}, \text{CQ})$, with $q = \exists \bar{y} \phi(x_1, \dots, x_n, \bar{y})$. Q can be equivalently rewritten as a (TGD, ACQ) query. More precisely, Q is equivalent to the query

$$Q' = (\mathbf{S} \cup \{P, P^*\}, \Sigma_{P^*} \cup \Sigma \cup \{\sigma_q\}, \text{Ans}(x_1, \dots, x_n)),$$

where $P/1, P^*/n$ are auxiliary predicates not in $\mathbf{S} \cup \text{sch}(\Sigma)$, Σ_{P^*} consists of the tgds:

$$\begin{aligned} R(x_1, \dots, x_n) &\rightarrow P(x_i), \text{ for each } R \in \mathbf{S} \text{ and } i \in \{1, \dots, n\} \\ P(x_1), \dots, P(x_n) &\rightarrow P^*(x_1, \dots, x_n), \end{aligned}$$

and σ_q is the tgd

$$P^*(x_1, \dots, x_n), \phi(x_1, \dots, x_n, \bar{y}) \rightarrow \text{Ans}(x_1, \dots, x_n).$$

In particular, Σ_{P^*} defines the predicate P^* that holds all the n -tuples over constants of the active domain, which then can be used in σ_q that converts the CQ q into a frontier-guarded tgd. Notice that Q' is not a query over a product database, which means we do not have access to the built-in predicate Dom^n . Therefore, in order to convert q into a frontier-guarded tgd, we need to explicitly construct all the n -tuples over the active domain and store them in the auxiliary predicate P^* . Although the set of tgds $\Sigma' = \Sigma_{P^*} \cup \Sigma \cup \{\sigma_q\}$ is not frontier-guarded, it has a very special form that allows us to rewrite it into a Datalog program by applying the translation Ξ . Observe that Σ' admits a stratification, where the first stratum is the set Σ_{P^*} , while the second stratum is the frontier-guarded set $\Sigma \cup \{\sigma_q\}$. This implies that Q' is equivalent to the Datalog query $(\Sigma_{P^*} \cup \Xi(\Sigma \cup \{\sigma_q\}), \text{Ans})$ over \mathbf{S} , and the claim follows.

It remains to show that $\text{DAT} \not\preceq (\text{FG}, \text{CQ})$. To this end, it suffices to construct a Datalog query Q over a schema \mathbf{S} such that, for every query $Q' \in (\text{FG}, \text{CQ})$ over \mathbf{S} , there exists an \mathbf{S} -database D such that, $Q(D) \neq Q'(D)$. We claim that such a Datalog query is Q_{trans} given in Example 1, which computes the transitive closure of the binary relation E . Towards a contradiction, assume that Q_{trans} can be expressed as (FG, CQ)

⁴ A similar result can be found in [4].

query (S, Σ, q) . Observe that frontier-guarded tgds are not able to put together in an atom, during the construction of the chase instance, two database constants that do not already coexist in a database atom. In particular, given a database D and a set $\Sigma \in \text{FG}$, if there is no atom in D that contains the constants $c, d \in \text{adom}(D)$, then there is no atom in $\text{chase}(D, \Sigma)$ that contains c and d . Therefore, q is able to compute the transitive closure of the binary relation E . But this contradicts the fact that a (finite) conjunctive query cannot compute the transitive closure of a binary relation. \square

We now show that product databases have an impact on the expressiveness of the ontology-mediated query languages based on (frontier-)guarded tgds. In fact, these languages become equally expressive to Datalog when we focus on product databases.

Lemma 3. $(G, CQ)_\times = (FG, CQ)_\times = \text{DAT}$

Proof. First observe that $(FG, CQ)_\times = (FG, ACQ)_\times$; recall that ACQ is the class of queries definable by some atomic CQ. More precisely, a query $(S, \Sigma, q) \in (FG, CQ)_\times$, with $q = \exists \bar{y} \phi(x_1, \dots, x_n, \bar{y})$, is equivalent to the $(FG, ACQ)_\times$ query

$$(S, \Sigma \cup \{\sigma_q\}, \text{Ans}(x_1, \dots, x_n)),$$

where σ_q is the tgd

$$\text{Dom}^n(x_1, \dots, x_n), \phi(x_1, \dots, x_n, \bar{y}) \rightarrow \text{Ans}(x_1, \dots, x_n),$$

which implies that $(FG, CQ)_\times \preceq (FG, ACQ)_\times$; the other direction holds trivially. Therefore, to prove our claim, it suffices to show that

$$(G, CQ)_\times \stackrel{(1)}{\preceq} (FG, ACQ)_\times \stackrel{(2)}{\preceq} \text{DAT} \stackrel{(3)}{\preceq} (G, CQ)_\times.$$

For showing (1), we observe that the construction given above for rewriting a $(FG, CQ)_\times$ query into a $(FG, ACQ)_\times$ query can be used in order to rewrite a $(G, CQ)_\times$ query into a $(FG, ACQ)_\times$ query. For showing (2), we can apply the procedure Ξ mentioned above, which transforms a (FG, ACQ) query into an equivalent DAT query. Finally, (3) follows from the fact that a Datalog rule ρ can be converted into a guarded tgd by adding in the body of ρ the atom $\text{Dom}^{|\bar{x}|}(\bar{x})$, where \bar{x} are the variables in ρ . \square

The next lemma shows that weakly-guarded sets of tgds give rise to an ontology-mediated query language that is strictly more expressive than Datalog.

Lemma 4. $\text{DAT} \prec (\text{WG}, CQ)$

Proof. $\text{DAT} \preceq (\text{WG}, CQ)$ holds trivially since a set of Datalog rules is a weakly-guarded set of tgds. In particular, a Datalog query $(\Sigma, \text{Ans}/n)$ over S is equivalent to the query $(S, \Sigma, \text{Ans}(x_1, \dots, x_n))$, where Σ is trivially weakly-guarded since there are no existentially quantified variables, which in turn implies that the set of affected positions of $\text{sch}(\Sigma)$ is empty. It remains to show that $(\text{WG}, CQ) \not\preceq \text{DAT}$. To this end, we employ a complexity-theoretic argument. It is well-known that the (decision version of the) problem of evaluating a Datalog query is feasible in polynomial time in data complexity, while for (WG, CQ) is complete for EXPTIME [7]. Thus, $(\text{WG}, CQ) \preceq \text{DAT}$ implies that $\text{PTIME} = \text{EXPTIME}$, which is a contradiction. \square

Class \mathcal{C}	Data Complexity	Bounded Arity	Combined Complexity
G	PTIME	EXPTIME	2EXPTIME
FG	PTIME	2EXPTIME	2EXPTIME
WG	EXPTIME	EXPTIME	2EXPTIME
WFG	EXPTIME	2EXPTIME	2EXPTIME

Table 1. Complexity of $\text{EVAL}((\mathcal{C}, \text{CQ})_{\times})$; all the results are completeness results.

We finally show that there is no impact on the expressiveness of the query languages that are based on weakly-(frontier-)guarded sets of tgds:

Lemma 5. $(\text{WG}, \text{CQ}) = (\text{WFG}, \text{CQ}) = (\text{WG}, \text{CQ})_{\times} = (\text{WFG}, \text{CQ})_{\times}$.

Proof. It is well-known that $(\text{WG}, \text{CQ}) = (\text{WFG}, \text{CQ})$; $(\text{WG}, \text{CQ}) \preceq (\text{WFG}, \text{CQ})$ holds trivially since $\text{WG} \subseteq \text{WFG}$, while $(\text{WFG}, \text{CQ}) \preceq (\text{WG}, \text{CQ})$ has been shown in [9]. It remains to show $(\text{WG}, \text{CQ}) = (\text{WG}, \text{CQ})_{\times}$ and $(\text{WFG}, \text{CQ}) = (\text{WFG}, \text{CQ})_{\times}$. The (\preceq) direction is trivial. The other direction holds since WG and WFG have the power to explicitly define a predicate P^k/k , where $k > 0$, that holds all the k -tuples of constants in the active domain. More precisely, a $(\text{WG}, \text{CQ})_{\times}$ (resp., $(\text{WFG}, \text{CQ})_{\times}$) query $Q = (\mathbf{S}, \Sigma, q)$, with $\alpha = \{j \mid \text{Dom}^j \in \mathbf{S}\}$, is equivalent to the (WG, CQ) (resp., (WFG, CQ)) query $Q' = (\mathbf{S}', \Sigma', q')$, where $\mathbf{S}' = \mathbf{S} \setminus \{\text{Dom}^k \mid k \in \alpha\}$, Σ' is obtained from Σ by replacing each predicate Dom^k with P^k and adding the set of tgds:

$$\begin{aligned} R(x_1, \dots, x_n) &\rightarrow P^1(x_1), \dots, P^1(x_n), \text{ for each } R \in \mathbf{S}' \\ P^1(x_1), \dots, P^1(x_k) &\rightarrow P^k(x_1, \dots, x_k), \text{ for each } k \in \alpha, \end{aligned}$$

and finally q' is obtained from q by replacing each predicate Dom^k with P^k . □

It is now easy to verify that Lemmas 1, 2, 3, 4 and 5 imply Theorem 1.

6 Complexity of Query Evaluation

The question that remains to be answered is whether product databases have an impact on the complexity of the query evaluation problem under the guarded-based ontology-mediated query languages in question. As is customary when studying the computational complexity of the evaluation problem for a query language, we consider its associated decision problem. We denote this problem by $\text{EVAL}(\mathcal{Q})$, where \mathcal{Q} is an ontology-mediated query language, and its definition follows:

INPUT : Query $Q = (\mathbf{S}, \Sigma, q(\bar{x})) \in \mathcal{Q}$, \mathbf{S} -database D , and tuple $\bar{t} \in \mathbf{C}^{|\bar{x}|}$.
 QUESTION : Does $\bar{t} \in \text{cert}_{q, \Sigma}(D)$?

It is important to say that when we focus on ontology-mediated queries over a product database, then the input database to the evaluation problem is product. In other words, if we focus on the problem $\text{EVAL}((\mathcal{C}, \text{CQ})_{\times})$, where \mathcal{C} is a class of tgds, and the input query is (\mathbf{S}, Σ, q) , then the input database is an α -product database, where $\alpha =$

$\{i \mid \text{Dom}^i \in \mathbf{S}\}$. The complexity of $\text{EVAL}((\mathcal{C}, \text{CQ}))$, where $\mathcal{C} \in \{\text{G}, \text{FG}, \text{WG}, \text{WFG}\}$, is well-understood; for (G, CQ) and (WG, CQ) it has been investigated in [7], while for (FG, CQ) and (WFG, CQ) in [3]. It is clear that the algorithms devised in [3, 7] for the guarded-based ontology-mediated query languages in question treat product databases in the same way as non-product databases, or, in other words, they are oblivious to the fact that an input database is product. Therefore, we can conclude that, even if we focus on product databases, the existing algorithms can be applied and get the same complexity results for query evaluation as in the case where we consider arbitrary (non-product) databases; these results are summarized in Table 1. Recall that the data complexity is calculated by considering only the database as part of the input, while in the combined complexity both the query and the database are part of the input. We also consider the important case where the arity of the schema is bounded by an integer constant.

6.1 The Bounded Arity Case Revisited

In Table 1, the bounded arity column refers to the case where all predicates in the given query, including the predicates of the form Dom^k , where $k > 0$, are of bounded arity. However, bounding the arity of the Dom^k predicates is not our intention. Observe that in the proof of Lemma 3, where we show $(\text{G}, \text{CQ})_{\times} = (\text{FG}, \text{CQ})_{\times} = \text{DAT}$, the predicates of the form Dom^k are used (i) to convert a CQ into a frontier-guarded tgd, and (ii) to convert a Datalog rule into a guarded tgd. More precisely, in the first case we use a Dom^k atom to guard the answer variables of a CQ, while in the second case to guard the variables in the body of a Datalog rule. Therefore, in both cases, we need to guard via a Dom^k atom an unbounded number of variables, even if the arity of the schema is bounded, and thus k must be unbounded. From the above discussion, it is clear that the interesting case to consider in our complexity analysis is not when all predicates of the underlying schema are of bounded arity, but when all predicates except the domain predicates are of bounded arity. Clearly, in case of $(\text{FG}, \text{CQ})_{\times}$ and $(\text{WFG}, \text{CQ})_{\times}$, the complexity of query evaluation is 2EXPTIME-complete, since the problem is 2EXPTIME-hard even if all predicates (including the domain predicates) have bounded arity. However, the picture is foggy in the case of $(\text{G}, \text{CQ})_{\times}$ and $(\text{WG}, \text{CQ})_{\times}$ since the existing results imply a 2EXPTIME upper bound and an EXPTIME lower bound. Interestingly, as we discuss below, the complexity of query evaluation remains the same, i.e., EXPTIME-complete, even if the domain predicates have unbounded arity.

Theorem 2. $\text{EVAL}((\text{WG}, \text{CQ})_{\times})$ is EXPTIME-complete if the arity of the schema, excluding the predicates of the form Dom^k , for $k > 0$, is bounded by an integer constant.

The lower bound follows from the fact $\text{EVAL}((\text{WG}, \text{CQ}))$ is EXPTIME-hard when the arity of the schema is bounded [7]. The upper bound relies on a result that, although is implicit in [7], it has not been explicitly stated before. The *body-predicates* of an ontology-mediated query (\mathbf{S}, Σ, q) are the predicates that do not appear in the head of a tgd of Σ . It holds that:

Proposition 1. $\text{EVAL}((\text{WG}, \text{CQ})_{\times})$ is in EXPTIME if the arity of the schema, excluding the body-predicates, is bounded by an integer constant.

The above result simply states that even if we allow the body-predicates to have unbounded arity, while all the other predicates of the schema are of bounded arity, the complexity of $\text{EVAL}((\text{WG}, \text{CQ})_\times)$ remains the same as in the case where all the predicates of the schema have bounded arity. Since the predicates of the form Dom^k , for $k > 0$, is a subset of the body-predicates of an ontology-mediated query over a product database, it is clear that Proposition 1 implies Theorem 2. As said, although Proposition 1 has not been explicitly stated before, it is implicit in [7], where the complexity of query evaluation for (WG, CQ) is investigated. In fact, we can apply the alternating algorithm devised in [7] for showing that $\text{EVAL}((\text{WG}, \text{CQ}))$ is in EXPTIME if the arity of the schema (including the body-predicates) is bounded by an integer constant. In what follows, we briefly recall the main ingredients of the alternating algorithm proposed in [7], and discuss how we get the desired upper bound.

Recall that a set $\Sigma \in \text{WG}$ can be effectively transformed into a set $\Sigma' \in \text{WG}$ such that all the tgds of Σ' are single-head [7]. Henceforth, for technical clarity, we focus on tgds with just one atom in the head. Let D be a database, and Σ a set of tgds. Fix a chase sequence $D = I_0 \xrightarrow{\sigma_0, h_0} I_1 \xrightarrow{\sigma_1, h_1} I_2 \dots$ for D under Σ . The instance $\text{chase}(D, \Sigma)$ can be naturally represented as a labeled directed graph $G = (N, E, \lambda)$ as follows: (1) for each atom $R(\bar{t}) \in \text{chase}(D, \Sigma)$, there exists $v \in N$ such that $\lambda(v) = R(\bar{t})$; (2) for each $i \geq 0$, with $I_i \xrightarrow{\sigma_i, h_i} I_{i+1}$, and for each atom $R(\bar{t}) \in h_i(\text{body}(\sigma_i))$, there exists $(v, u) \in E$ such that $\lambda(v) = R(\bar{t})$ and $\{\lambda(u)\} = I_{i+1} \setminus I_i$; and (3) there are no other nodes and edges in G . The *guarded chase forest* of D and Σ , denoted $\text{gcf}(D, \Sigma)$, is the forest obtained from G by keeping only the nodes associated with weak-guards, and their children; for more details, we refer the reader to [7].

Consider a query $(\mathbf{S}, \Sigma, q) \in (\text{WG}, \text{CQ})$, a database D over \mathbf{S} , and a tuple \bar{t} of constants. Clearly, $\bar{t} \in \text{cert}_{q, \Sigma}(D)$ iff there exists a homomorphism that maps $q(\bar{t})$ to $\text{gcf}(D, \Sigma)$. Observe that if such a homomorphism h exists, then in $\text{gcf}(D, \Sigma)$ there exist paths starting from nodes labeled with database atoms and ending at nodes labeled with atom of $h(q(\bar{t}))$. The alternating algorithm in [7] first guesses the homomorphism h from $q(\bar{t})$ to $\text{gcf}(D, \Sigma)$, and then constructs in parallel universal computations the paths from D to $h(q(\bar{t}))$ (if they exist). During this alternating process, the algorithm exploits a key result established in [7], that is, the subtree of $\text{gcf}(D, \Sigma)$ rooted at some atom $R(\bar{u})$ is determined by the so-called cloud of $R(\bar{u})$ (modulo renaming of nulls) [7, Theorem 5.16]. The *cloud* of $R(\bar{u})$ with respect to D and Σ , denoted $\text{cloud}(R(\bar{u}), D, \Sigma)$, is defined as $\{S(\bar{v}) \in \text{chase}(D, \Sigma) \mid \bar{v} \subseteq (\text{adom}(D) \cup \bar{u})\}$, i.e., the atoms in the result of the chase with constants from D and terms from \bar{u} . This result allows the algorithm to build the relevant paths of $\text{gcf}(D, \Sigma)$ from D to $h(q(\bar{t}))$. Roughly, an atom $R(\bar{u})$ on a path can be generated by considering only its parent atom $S(\bar{v})$ and the cloud of $S(\bar{v})$ with respect to D and Σ . Whenever a new atom is generated, the algorithm nondeterministically guesses its cloud, and verify in a parallel universal computation that indeed belongs to the result of the chase.

From the above informal description, we conclude that the space needed at each step of the computation of the alternating algorithm is actually the size of the cloud of an atom. By applying a simple combinatorial argument, it is easy to show that the size of a cloud is at most $(|\mathbf{S}| + |\text{sch}(\Sigma)|) \cdot (|\text{adom}(D)| + w)^w$, where w is the maximum arity over all predicates of $\mathbf{S} \cup \text{sch}(\Sigma)$. Therefore, if we assume that all the predicates

of the schema have bounded arity, which means that w is a constant, then the size of a cloud is polynomial. Since alternating polynomial space coincides with deterministic exponential time, we immediately get the EXPTIME upper bound in the case of bounded arity. Now, let \mathbf{B} be the body-predicates of (\mathbf{S}, Σ, q) . It is clear that, for every atom $R(\bar{u}) \in \text{chase}(D, \Sigma)$, the restriction of $\text{cloud}(R(\bar{u}), D, \Sigma)$ on the predicates of \mathbf{B} is actually D , and thus of polynomial size, even if the predicates of \mathbf{B} have unbounded arity. This implies that even if we allow body-predicates of unbounded arity the size of a cloud remains polynomial. Therefore, the alternating algorithm devised in [7] can be applied in order to get the EXPTIME upper bound stated in Proposition 1.

7 Conclusions

It is realistic to assume that a database management system provides access to the active domain via built-in relations, or, in more formal terms, to assume that queries are evaluated over product databases. Interestingly, the query languages that are based on (frontier-)guarded existential rules gain in expressiveness when we focus on product databases; in fact, they have the same expressive power as Datalog. On the other hand, there is no impact on the expressive power of the query languages based on weakly-(frontier-)guarded existential rules, since they are powerful enough to explicitly compute the predicates needed to access the active domain. We also observe that there is no impact on the computational complexity of the query languages in question.

Acknowledgements: Gottlob is supported by the EPSRC Programme Grant EP/M025268/. Pieris and Šimkus are supported by the Austrian Science Fund (FWF), projects P25207-N23 and Y698.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: Walking the decidability line. *Artif. Intell.* 175(9-10), 1620–1654 (2011)
3. Baget, J.F., Mugnier, M.L., Rudolph, S., Thomazo, M.: Walking the complexity lines for generalized guarded existential rules. In: *IJCAI*. pp. 712–717 (2011)
4. Bárány, V., Benedikt, M., ten Cate, B.: Rewriting guarded negation queries. In: *MFCS*. pp. 98–110 (2013)
5. Beeri, C., Vardi, M.Y.: The implication problem for data dependencies. In: *ICALP*. pp. 73–85 (1981)
6. Bienvenu, M., ten Cate, B., Lutz, C., Wolter, F.: Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.* 39(4), 33:1–33:44 (2014)
7. Cali, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.* 48, 115–174 (2013)
8. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. *Theor. Comput. Sci.* 336(1), 89–124 (2005)
9. Gottlob, G., Rudolph, S., Simkus, M.: Expressiveness of guarded existential rule languages. In: *PODS*. pp. 27–38 (2014)
10. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.* 28(1), 167–189 (1984)
11. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. *ACM Trans. Database Syst.* 4(4), 455–469 (1979)