Dyadic Existential Rules

GEORG GOTTLOB

Department of Computer Science, University of Oxford, Oxford OX1 3QG, UK
Faculty of Informatics, TU Wien, Vienna, Austria
(*e-mail:* georg.gottlob@cs.ox.ac.uk)

MARCO MANNA and CINZIA MARTE

Department of Mathematics and Computer Science, University of Calabria, Arcavacata, Italy
(*e-mails:* marco.manna@unical.it, cinzia.marte@unical.it)

Abstract

Existential rules form an expressive Datalog-based language to specify ontological knowledge. The presence of existential quantification in rule-heads, however, makes the main reasoning tasks undecidable. To overcome this limitation, in the last two decades, a number of classes of existential rules guaranteeing the decidability of query answering have been proposed. Unfortunately, only some of these classes fully encompass Datalog and, often, this comes at the price of higher computational complexity. Moreover, expressive classes are typically unable to exploit tools developed for classes exhibiting lower expressiveness. To mitigate these shortcomings, this paper introduces a novel general syntactic condition that allows us to define, systematically and in a uniform way, from any decidable class $\mathcal{C}$ of existential rules, a new class called Dyadic-$\mathcal{C}$ enjoying the following properties: (i) it is decidable; (ii) it generalizes Datalog; (iii) it generalizes $\mathcal{C}$; (iv) it can effectively exploit any reasoner for query answering over $\mathcal{C}$; and (v) its computational complexity does not exceed the highest between the one of $\mathcal{C}$ and the one of Datalog.

KEYWORDS: existential rules, Datalog, ontology-based query answering, tuple-generating dependencies, computational complexity

1 Introduction

In ontology-based query answering, a conjunctive query is typically evaluated over a logical theory consisting of a relational database paired with an ontology. Description logics (Baader *et al.* 2003) and existential rules – a.k.a. tuple generating dependencies, or Datalog$^\pm$ rules – (Baget *et al.* 2011) are the main languages used to specify ontologies. In particular, the latter are essentially classical datalog rules (Abiteboul *et al.* 1995) extended with existential quantified variables in rule-heads. The presence of existential quantification in the head of rules, however, makes query answering undecidable in the general case. To overcome this limitation, in the last two decades, a number of classes of existential rules – based on both semantic and syntactic conditions – that guarantee the decidability of query answering have been proposed. Concerning the semantic conditions, we recall *finite expansions sets*, *finite treewidth sets*, *finite unification sets*, and *strongly parsimonious sets* (Baget *et al.* 2009; Baget *et al.* 2011; Leone *et al.* 2019). Each of

Table 1. *Computational complexity of query answering*

Class $\mathcal{C}$	Data complexity	Combined complexity
Weakly-(Fr)-Guarded[1],[2]	**EXPTIME**-complete	**2EXPTIME**-complete
(Fr)-Guarded[2]	**PTIME**-complete	**2EXPTIME**-complete
Weakly-Acyclic[3]	**PTIME**-complete	**2EXPTIME**-complete
Jointly-Acyclic[4]	**PTIME**-complete	**2EXPTIME**-complete
Datalog[5]	**PTIME**-complete	**EXPTIME**-complete
Shy[6]	**PTIME**-complete	**EXPTIME**-complete
Ward[7]	**PTIME**-complete	**EXPTIME**-complete
Protected[8]	**PTIME**-complete	**EXPTIME**-complete
Sticky-(Join)[9]	$\mathbf{AC_0}$	**EXPTIME**-complete
Linear[10]	$\mathbf{AC_0}$	**PSPACE**-complete
Joinless[11]	$\mathbf{AC_0}$	**PSPACE**-complete
Inclusion-Dependencies[12]	$\mathbf{AC_0}$	**PSPACE**-complete

[1] Baget *et al.* (2011); [2] Calì *et al.* (2013); [3] Fagin *et al.* (2005); [4] Krötzsch and Rudolph (2011); [5] Ceri *et al.* (1989); [6] Leone *et al.* (2019); [7] Gottlob and Pieris (2015); [8] Baldazzi *et al.* (2022); [9] Calì *et al.* (2012b); [10] Calì *et al.* (2012a); [11] Gogacz and Marcinkowski (2017); [12] Johnson and Klug (1984).

these classes encompasses a number of concrete classes based on syntactic conditions (Baget *et al.* 2011; Calì *et al.* 2013; Fagin *et al.* 2005; Krötzsch and Rudolph 2011; Ceri *et al.* 1989; Leone *et al.* 2019; Gottlob and Pieris 2015; Baldazzi *et al.* 2022; Calì *et al.* 2012b; Calì *et al.* 2012a; Gogacz and Marcinkowski 2017; Johnson and Klug 1984). Table 1 summarizes these classes and their computational complexity with respect to query answering, by distinguishing between *combined complexity* (the input consists of a database, an ontology, a conjunctive query, and a tuple of constants) and *data complexity* (only a database is given as input, whereas the remaining parameters are considered fixed).

Unfortunately, on the one side, despite the fact that existential rules generalize datalog rules, only some of these syntactic classes fully encompass Datalog and, in some cases, this even comes at the price of higher computational complexity of query answering. Moreover, on the other side, expressive classes typically need *ad hoc* reasoners without being able to exploit mature tools developed for classes exhibiting lower expressiveness.

With the aim of mitigating the two aforementioned shortcomings, this paper introduces a novel general syntactic condition that allows to define, systematically and in a uniform way, from any decidable class $\mathcal{C}$ of existential rules, a new class called Dyadic-$\mathcal{C}$ that enjoys the following properties: (*i*) it is decidable; (*ii*) it generalizes Datalog;[1] (*iii*) it generalizes $\mathcal{C}$; and (*iv*) it can effectively exploit any reasoner for query answering over $\mathcal{C}$. In particular, let $\mathbb{C}_d$ (resp., $\mathbb{C}_c$) be the data (resp., combined) complexity of query answering over $\mathcal{C}$,

---

[1] Strictly speaking, to guarantee that Dyadic-$\mathcal{C}$ generalizes Datalog, one has to focus on any $\mathcal{C} \supseteq$ Af-Inds, where Af-Inds is the very simple class of existential rules defined in Section 3.3 such that (*a*) rules are inclusion dependencies with no existential variable and (*b*) predicates in rule-heads do not appear in rule-bodies. Indeed, all known classes of existential rules based on semantic conditions as well as all concrete classes reported in Table 1 do encompass Af-Inds.

query answering over Dyadic-$\mathcal{C}$ is $\textbf{PTIME}^{\mathbb{C}_d}$ (resp., $\textbf{EXPTIME}^{\mathbb{C}_c}$ provided that there is at least an exponential jump from $\mathbb{C}_d$ to $\mathbb{C}_c$). Since all the classes reported in Table 1 comply with the exponential jump assumption, we get the following: (a) whenever $\mathbb{C}_d \supseteq \textbf{PTIME}$ (entries 1–8 of Table 1), then query answering over Dyadic-$\mathcal{C}$ is complete for $\mathbb{C}_d$ (resp., $\mathbb{C}_c$); (b) in all the remaining cases (entries 9-12 of Table 1), query answering over Dyadic-$\mathcal{C}$ is complete for $\textbf{PTIME}$ (resp., $\textbf{EXPTIME}$), namely it has the same complexity of query answering over Datalog.

Concerning the key principle at the heart of this new general syntactic condition, basically, an ontology Σ belongs to Dyadic-$\mathcal{C}$ if one can easily construct a pair $(\Sigma_{\text{HG}}, \Sigma_{\mathcal{C}})$ of ontologies, called *dyadic*, such that: (i) $\Sigma_{\text{HG}} \cup \Sigma_{\mathcal{C}}$ is equivalent to Σ with respect to query answering; (ii) $\Sigma_{\mathcal{C}} \in \mathcal{C}$; and (iii) Σ_{HG} is a set rules called *head-ground* with respect to $\Sigma_{\text{HG}} \cup \Sigma_{\mathcal{C}}$ (Gottlob and Pieris 2015). Intuitively, Σ_{HG} satisfies the following properties: (1) it belongs to Datalog; (2) for each database D, the chase procedure (Deutsch *et al.* 2008) over $D \cup \Sigma_{\text{HG}} \cup \Sigma_{\mathcal{C}}$ never generates atoms containing null-values via rules of Σ_{HG}; (3) head-predicates of Σ_{HG} and body-predicates of Σ_{HG} are disjoint; and (4) head-predicates of Σ_{HG} and head-predicates of $\Sigma_{\mathcal{C}}$ are disjoint. Finally, since Dyadic-$\mathcal{C}$ is well-defined even if $\mathcal{C}$ is a class of existential rules based on some semantic conditions and, if so, since query answering is still decidable over Dyadic-$\mathcal{C}$, then – in analogy with the existing semantic classes – the union of all the Dyadic-$\mathcal{C}$ classes are called *dyadic decomposable sets*.

The article is a revised version of an earlier workshop paper (Gottlob *et al.* 2022). Specifically, the content that was previously presented in a single preliminary section has been expanded and reorganized into two longer separate sections, namely Sections 2 and 3. These sections now contain the necessary background information, ensuring that the paper is self-contained. Furthermore, the previous notion of "dyadic decomposition" has evolved into the novel notion of a "Dyadic Pair of TGDs", which is discussed in Section 4. This new notion captures the essential properties of dyadic decompositions and also generalizes the notion of ontology, providing new perspectives and insights. Additionally, in Section 5, the notion of "Dyadic Decomposable Sets" is now supported by a canonical concrete algorithm that produces a Dyadic Pair of TGDs from each Dyadic Decomposable Set. The revisions also lead to new results regarding decidability and complexity. First, if $\mathcal{C}$ is an abstract (resp., concrete) and decidable class, then Dyadic-$\mathcal{C}$ is now also an abstract (resp., concrete) and decidable class. Second, the relationship between Datalog and any Dyadic-$\mathcal{C}$ is made explicit, emphasizing the low expressive power required for $\mathcal{C}$ to ensure that Dyadic-$\mathcal{C}$ fully encompasses Datalog. Finally, the computational complexity analysis is completed in Section 6, where both data and combined complexity for any Dyadic-$\mathcal{C}$ class are systematically studied.

2 Preliminaries

In this section, we introduce the syntax and the semantics of the class of rules that generalizes Datalog with existential quantifiers in rule-heads. Regarding computational complexity, we assume the reader is familiar with the basic complexity classes used in the subsequent sections: $\textbf{AC}_0 \subseteq \textbf{PTIME} \subseteq \textbf{NP} \subseteq \textbf{PSPACE} \subseteq \textbf{EXPTIME} \subseteq \textbf{2EXPTIME}$. Moreover, for a complexity class $\mathbb{C}$, we denote by $\textbf{PTIME}^{\mathbb{C}}$ (resp., $\textbf{EXPTIME}^{\mathbb{C}}$) the class of decision problems that can be solved by an oracle Turing machine operating in polynomial (resp., exponential) time with the aid of an oracle that decides a problem in $\mathbb{C}$.

2.1 Basics on relational structures

Fix three pairwise disjoint lexicographically enumerable infinite sets C of *constants*, N of *nulls* (φ, φ_0, φ_1, ...), and V of *variables* (x, y, z, and variations thereof). Their union is denoted by T and its elements are called *terms*. For any integer $k \geq 0$, we may write $[k]$ for the set $\{1, ..., k\}$; in particular, as usual, if $k = 0$, then $[k] = \emptyset$.

An *atom* $\underline{a}$ is an expression of the form $P(\mathbf{t})$, where $preds(\underline{a}) = P$ is a *(relational) predicate*, $\mathbf{t} = t_1, ..., t_k$ is a tuple of *terms* $arity(\underline{a}) = arity(P) = k \geq 0$ is the *arity* of both $\underline{a}$ and P, and $\underline{a}[i]$ denotes the i-th term $\mathbf{t}[i] = t_i$ of $\underline{a}$, for each $i \in [k]$. In particular, if $k = 0$, then $\mathbf{t}$ is the empty tuple and $\underline{a} = P()$. By $consts(\underline{a})$ and $vars(\underline{a})$ we denote, respectively, the set of constants and variables occurring in $\underline{a}$. A *fact* is an atom that contains only constants.

A *(relational) schema* $\mathbf{S}$ is a finite set of predicates, each with its own arity. The set of *positions* of $\mathbf{S}$, denoted by $pos(\mathbf{S})$, is defined as the set $\{P[i] \mid P \in \mathbf{S} \wedge 1 \leq i \leq arity(P)\}$, where each $P[i]$ denotes the i-th *position* of P. A *(relational) structure* over $\mathbf{S}$ is any (possibly infinite) set of atoms using only predicates from $\mathbf{S}$. The *domain* of a structure S, denoted by $dom(S)$, is the set of all the terms forming the atoms of S. An *instance* over $\mathbf{S}$ is any structure I over $\mathbf{S}$ such that $dom(I) \subseteq \mathsf{C} \cup \mathsf{N}$. A *database* over $\mathbf{S}$ is any finite instance over $\mathbf{S}$ containing only facts. The *active domain* of an instance I, denoted by $dom(I)$, is the set of all the terms occurring in I, whereas the *Herbrand Base* of I, denoted by $HB(I)$, is the set of all the atoms that can be formed using the predicate symbols of $\mathbf{S}$ and terms of $dom(I)$.

Consider two sets of terms T_1 and T_2 and a map $\mu : T_1 \rightarrow T_2$. Given a set T of terms, the *restriction* of μ with respect to T is the map $\mu|_T = \{t \mapsto \mu(t) : t \in T_1 \cap T\}$. An *extension* of μ is any map μ' between terms, denoted by $\mu' \supseteq \mu$, such that $\mu'|_{T_1} = \mu$. A *homomorphism* from a structure S_1 to a structure S_2 is any map $h : dom(S_1) \rightarrow dom(S_2)$ such that both the following hold: (i) if $t \in \mathsf{C} \cap dom(S_1)$, then $h(t) = t$; and (ii) $h(S_1) = \{P(h(\mathbf{t})) : P(\mathbf{t}) \in S_1\} \subseteq S_2$.

2.2 Conjunctive queries

A *conjunctive query* (CQ) q over a schema $\mathbf{S}$ is a (first-order) formula of the form

$$\langle \mathbf{x} \rangle \leftarrow \exists \, \mathbf{y} \; \Phi(\mathbf{x}, \mathbf{y}), \tag{1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are tuples (often seen as sets) of variables such that $\mathbf{x} \cap \mathbf{y} = \emptyset$, and $\Phi(\mathbf{x}, \mathbf{y})$ is a conjunction (often seen as a set) of atoms using only predicates from $\mathbf{S}$. In particular,

- $dom(\Phi) \subseteq \mathbf{x} \cup \mathbf{y} \cup \mathsf{C}$,
- whenever a variable z belongs to $\mathbf{x} \cup \mathbf{y}$, then z occurs also in Φ,
- $\mathbf{x}$ are the *output* variables of q, and
- $\mathbf{y}$ are the *existential* variables of q.

To highlight the output variables, we may write $q(\mathbf{x})$ instead of q. The *evaluation* of q over an instance I is the set $q(I)$ of every tuple $\mathbf{t}$ of constants admitting a homomorphism $h_{\mathbf{t}}$ from $\Phi(\mathbf{x}, \mathbf{y})$ to I such that $h_{\mathbf{t}}(\mathbf{x}) = \mathbf{t}$.

A *Boolean conjunctive query* (BCQ) is a CQ with no output variable, namely an expression of the form $\langle \rangle \leftarrow \exists \, \mathbf{y} \; \Phi(\mathbf{y})$. An instance I satisfies a BCQ q, denoted $I \models q$, if $q(I)$ is nonempty, namely $q(I)$ contains only the empty tuple $\langle \rangle$.

2.3 Tuple-generating dependencies

A *tuple-generating dependency* (TGD) σ – also known as *(existential) rule* – over a schema **S** is a (first-order) formula of the form

$$\Phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists\, \mathbf{z}\ \Psi(\mathbf{x}, \mathbf{z}), \tag{2}$$

where $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ are pairwise disjoint tuples of variables, and both $\Phi(\mathbf{x}, \mathbf{y})$ and $\Psi(\mathbf{x}, \mathbf{z})$ are conjunctions (often seen as a sets) of atoms using only predicates from **S**. In particular,

- Φ (resp., Ψ) contains all and only the variables in $\mathbf{x} \cup \mathbf{y}$ (resp., $\mathbf{x} \cup \mathbf{z}$),
- constants (but not nulls) may also occur in σ,
- $\mathbf{x} \cup \mathbf{y}$ are the *universal* variables of σ denoted by $vars_\forall(\sigma)$,
- $\mathbf{z}$ are the *existential* variables of σ denoted by $vars_\exists(\sigma)$, and
- $\mathbf{x}$ are the *frontier* variables of σ denoted by $\mathsf{vars}_\frown(\sigma)$.

We refer to $body(\sigma) = \Phi$ and $head(\sigma) = \Psi$ as the *body* and *head* of σ, respectively. If $|head(\sigma)| = 1$, the TGD is called *single-head*, otherwise it called *multi-head*. If $vars_\exists(\sigma) = \emptyset$ and $|head(\sigma)| = 1$, then σ is called *datalog rule*. With $h\text{-}preds(\sigma)$ (resp., $b\text{-}preds(\sigma)$) we denote the set of predicates in $head(\sigma)$ (resp., $body(\sigma)$). An instance I satisfies σ, written $I \models \sigma$, if the existence of a homomorphism h from Φ to I implies the existence of a homomorphism $h' \supseteq h_{|\mathbf{x}}$ from Ψ to I.

An *ontology* Σ is a set of rules. An instance I satisfies Σ, written $I \models \Sigma$, if $I \models \sigma$ for each $\sigma \in \Sigma$. Without loss of generality, we assume that $vars(\sigma_1) \cap vars(\sigma_2) = \emptyset$, for each pair σ_1, σ_2 of rules in Σ. Operators $vars_\exists$, $h\text{-}preds$, and $b\text{-}preds$ naturally extend on ontologies.

A class $\mathcal{C}$ of ontologies is any (typically infinite) set of TGDs fulfilling some syntactic or semantic conditions (see, e.g., the classes shown in Table 1, some of which will be formally defined in the subsequent sections). In particular, Datalog is the class of ontologies containing only datalog rules.

Finally, the *schema* of an ontology Σ, denoted $sch(\Sigma)$, is the subset of **S** containing all and only the predicates occurring in Σ, whereas $arity(\Sigma) = \max_{P \in sch(\Sigma)} arity(P)$. For simplicity of exposition, we write $pos(\Sigma)$ instead of $pos(sch(\Sigma))$.

2.4 Ontological query answering

Consider a database D and a set Σ of TGDs. A *model* of D and Σ is an instance I such that $I \supseteq D$ and $I \models \Sigma$. Let $mods(D, \Sigma)$ be the set of all models of D and Σ. The *certain answers* to a CQ q w.r.t. D and Σ are defined as the set of tuples $cert(q, D, \Sigma) = \bigcap_{M \in mods(D, \Sigma)} q(M)$. Accordingly, for any fixed schema **S**, two ontologies Σ_1 and Σ_2 over **S** are said to be **S**-*equivalent* (in symbols $\Sigma_1 \equiv_\mathbf{S} \Sigma_2$) if, for each D and q over **S**, it holds that $cert(q, D, \Sigma_1) = cert(q, D, \Sigma_2)$. The pair D and Σ satisfies a BCQ q, written $D \cup \Sigma \models q$, if $cert(q, D, \Sigma) = \langle\rangle$, namely $M \models q$ for each $M \in mods(D, \Sigma)$. Fix a class $\mathcal{C}$ of ontologies. The computational problem studied in this work – called CERT-EVAL[$\mathcal{C}$] – can be schematized as follows:

PROBLEM: CERT-EVAL[$\mathcal{C}$].
    INPUT: A database D, a ontology $\Sigma \in \mathcal{C}$, a conjunctive query $q(\mathbf{x})$, and a tuple $\mathbf{c} \in \mathsf{C}^{|\mathbf{x}|}$.
QUESTION: Does $\mathbf{c} \in cert(q, D, \Sigma)$ hold?

In what follows, with a slight abuse of terminology, whenever we say that $\mathcal{C}$ is decidable, we mean that CERT-EVAL[$\mathcal{C}$] is decidable. Note that $\mathbf{c} \in cert(q, D, \Sigma)$ if, and only if, $D \cup \Sigma \models q(\mathbf{c})$, where $q(\mathbf{c})$ is the BCQ obtained from $q(\mathbf{x})$ by replacing, for each $i \in \{1, ..., |\mathbf{x}|\}$, every occurrence of the variable $\mathbf{x}[i]$ with the constant $\mathbf{c}[i]$. Actually, the former problem is $\mathbf{AC}_0$ reducible to the latter.

While considering the computational complexity of CERT-EVAL[$\mathcal{C}$], we recall the following convention: (*i*) *combined complexity* means that D, Σ, q, and $\mathbf{c}$ are given in input; and (*ii*) *data complexity* means that only D and $\mathbf{c}$ are given in input, whereas Σ and q are considered fixed. Accordingly, we point out that complexity results reported in Table 1 refer to CERT-EVAL[$\mathcal{C}$] under this convention.

2.5 The chase procedure

The chase procedure (Deutsch *et al.* 2008) is a tool exploited for reasoning with TGDs. Consider a database D and a set Σ of TGDs. Given an instance $I \supseteq D$, a *trigger* for I is any pair $\langle \sigma, h \rangle$, where $\sigma \in \Sigma$ is a rule as in equation (2) and h is a homomorphism from $body(\sigma)$ to I. Let $I' = I \cup h'(head(\sigma))$, where $h' \supseteq h|_{\mathbf{x}}$ maps each $z \in vars_{\exists}(\sigma)$ to a "fresh" null $h'(z)$ not occurring in I such that $z_1 \neq z_2$ in $vars_{\exists}(\sigma)$ implies $h'(z_1) \neq h'(z_2)$. Such an operation which constructs I' from I is called *chase step* and denoted $\langle \sigma, h \rangle(I) = I'$.

Without loss of generality, we assume that nulls introduced at each trigger functionally depend on the pair $\langle \sigma, h \rangle$ that is involved in the trigger. For example, given a rule σ as in equation (2) and a homomorphism h, it is sufficient to pick $\varphi_{\langle \mathbf{z}, h(\mathbf{x}, \mathbf{y}) \rangle}$ as the fresh null replacing $\mathbf{z}$ when the chase produces the trigger $\langle \sigma, h \rangle$. Accordingly, the processing order of rules and triggers does not change the result of the chase, and hence $chase(D, \Sigma)$ can be considered unique. The *chase procedure* of $D \cup \Sigma$ is an exhaustive application of chase steps, starting from D, which produce a sequence $I_0 = D \subset I_1 \subset I_2 \subset \cdots \subset I_m \subset \ldots$ of instances in such a way that: (*i*) for each $i \geq 0$, $I_{i+1} = \langle \sigma, h \rangle(I_i)$ is a chase step obtained via some trigger $\langle \sigma, h \rangle$ for I_i; (*ii*) for each $i \geq 0$, if there exists a trigger $\langle \sigma, h \rangle$ for I_i, then there exists some $j > i$ such that $I_j = \langle \sigma, h \rangle(I_{j-1})$ is a chase step; and (*iii*) any *trigger* $\langle \sigma, h \rangle$ is used only once. We define $chase(D, \Sigma) = \cup_{i \geq 0} I_i$.

The *chase bottom* is the finite set of all null-free atoms in $chase(D, \Sigma)$ and is defined as $chase^{\perp}(D, \Sigma) = chase(D, \Sigma) \cap HB(D)$.

It is well know that $chase(D, \Sigma)$ is a *universal model* of $D \cup \Sigma$, that is, for each $M \in mods(D, \Sigma)$ there is a homomorphism from $chase(D, \Sigma)$ to M. Hence, given a BCQ q it holds that $chase(D, \Sigma) \models q \Leftrightarrow D \cup \Sigma \models q$.

We recall that $chase(D, \Sigma)$ can be decomposed into *levels* (Calì *et al.* 2010): each atom of D has level $\gamma = 0$; an atom of $chase(D, \Sigma)$ has level $\gamma + 1$ if, during its generation, the exploited trigger $\langle \sigma, h \rangle$ maps the body of σ via h to atoms whose maximum level is γ. We refer to the part of the chase *up to* level γ as $chase^{\gamma}(D, \Sigma)$. Clearly, $chase(D, \Sigma) = \cup_{\gamma \geq 0} chase^{\gamma}(D, \Sigma)$. Finally, a trigger *involved* at a certain level j if it gives rise to an atom of level j.

3 Considered decidable classes of TGDs

In this section we provide an overview of the main existing decidable classes of TGDs. We recall both syntactic and semantic classes, where the first are based on a specific

syntactic condition that can be checked, while the latter are classes that do not come with a syntactic property that can be checked on rules and, hence, are not recognizable. Finally, we introduce a very simple new class of existential rules called Af-Inds. We will exploit the latter to sharpen our results presented in Sections 5 and 6.

3.1 Preliminary notions

We start fixing some basics notions. We have chosen to provide a uniform notation for the key existing notions of affected and invaded positions, such as attacked, protected, harmless, harmful, and dangerous variables (Leone *et al.* 2019; Calì *et al.* 2013; Krötzsch and Rudolph 2011; Berger *et al.* 2022; Gottlob *et al.* 2022). Basically, these notions serve to separate positions in which the chase can introduce only constants from those where nulls might appear.

Definition 1 (S-affected positions)
Consider an ontology Σ and a variable $z \in vars_\exists(\Sigma)$. A position $\pi \in \mathsf{pos}(\Sigma)$ is z-*affected* (or *invaded* by z) if one of the following two properties holds: (i) there exists $\sigma \in \Sigma$ such that z appears in the head of σ at position π; (ii) there exist $\sigma \in \Sigma$ and $x \in \mathsf{front}(\sigma)$ such that x occurs both in $head(\sigma)$ at position π and in $body(\sigma)$ at z-affected positions only. Moreover, a position $\pi \in \mathsf{pos}(\Sigma)$ is S-*affected*, where $S \subseteq vars_\exists(\Sigma)$, if: (i) for each $z \in S$, π is z-affected; and (ii) for each $z \in vars_\exists(\Sigma)$, if π is z-affected, then $z \in S$. ∎

Note that for every position π there exists a unique set S such that π is S-affected. We write $aff(\pi)$ for this set S. Moreover, $aff(\Sigma) = \{\pi \in pos(\Sigma) \mid aff(\pi) \neq \emptyset\}$, and $nonaff(\Sigma) = pos(\Sigma) \setminus aff(\Sigma)$.

We point out that the notion above presented is a refined version of the classical notion of affected position (Calì *et al.* 2008). In particular, it holds that if a position π is S-affected, then π is also affected; whereas if π is affected, then π may not be S-affected. Moreover, the S-affected notion coincides with the one of *attacked positions by a variable* (Leone *et al.* 2019; Krötzsch and Rudolph 2011). We highlight that its key nature and properties are not modified by the notion of S-affected position introduced above. Hence, for simplicity of exposition, we give only this refined definition. In the same spirit, we classify variables occurring in a conjunction of atoms.

Definition 2 (Variables classification)
Let σ be a TGD and $x \in vars(body(\sigma))$. Then, (i) if x occurs at positions $\pi_1, \ldots, \pi_n$ and $\bigcap_{i=1}^n \mathsf{aff}(\pi_i) = \emptyset$, x is said to be *harmless*; (ii) if x is not harmless, with $S = \bigcap_{i=1}^n \mathsf{aff}(\pi_i)$, it is said to be S-*harmful*; (iii) if x is S-harmful and belongs to $\mathsf{vars}_\frown(\sigma)$, x is S-*dangerous*. ∎

Given a variable x that is S-dangerous, we write $dang(x)$ for the set S. Hereinafter, for simplicity of exposition, the prefix S- is omitted when it is not necessary. Consider an ontology Σ. Given a rule $\sigma \in \Sigma$, we denote by $dang(\sigma)$ (resp., $harmless(\sigma)$ and $harmful(\sigma)$) the dangerous (resp., harmless and harmful) variables in σ. These sets of variables naturally extend to the whole Σ by taking, for each of them, the union over all the rules of Σ.

3.2 Decidable classes of existential rules

We now survey the 15 concrete classes reported in Table 1 as well as the known abstract classes based on semantic conditions. On the one side, we report some specific syntactic conditions whenever these are useful for the rest of the presentation; on the other side, for all of them (both concrete and abstract), we recall their containment relationships. For the rest of the section, fix a Datalog$^\exists$ ontology Σ.

The class FES (Baget *et al.* 2009) stands for finite expansions sets, which intuitively are sets of TGDs which ensure the termination of the chase procedure. The class BTS (Baget *et al.* 2009) stands for bounded treewidth sets, which intuitively are sets of TGDs which guarantee that the (possibly infinite) instance constructed by the chase procedure has bounded treewidth. The class FUS (Baget *et al.* 2011) stands for finite unification sets, which intuitively are sets of TGDs which guarantee the termination of (resolution-based) backward chaining procedures. The class SPS (Leone *et al.* 2019) stands for strongly parsimonious sets, which intuitively are sets of TGDs which guarantee that the parsimonious chase procedure can be reapplied a number of times that is linear in the size of the query.

The class Weakly-Acyclic (Fagin *et al.* 2005) is based on the *acyclicity* condition. To define the latter, we recall the *label graph* of Σ, $G(\Sigma) = \langle N, A \rangle$, defined as follows: (i) $N = \cup_{P \in sch(\Sigma)} pos(P)$; (ii) $(\pi_1, \pi_2, \forall) \in A$ if there are $\sigma \in \Sigma$ and $x \in \mathsf{vars}_\frown(\sigma)$ such that x occurs both in $body(\sigma)$ at position π_1 and in $head(\sigma)$ at position π_2; and (iii) $(\pi_1, \pi_2, \exists) \in A$ if there are $\sigma \in \Sigma$, $x \in \mathsf{vars}_\frown(\sigma)$, and $y \in vars_\exists(\sigma)$ such that both x occurs in $body(\sigma)$ at position π_1 and y occurs in $head(\sigma)$ at position π_2. The *existential graph* of Σ is $G_\exists(\Sigma) = \langle N, A \rangle$, where: (i) $N = \cup_{\sigma \in \Sigma} vars_\exists(\sigma)$; and (ii) $(X, Y) \in A$ if the rule σ where y occurs contains a universal variable x-affected and occurring in $head(\sigma)$. Therefore, Σ belongs to Weakly-Acyclic (resp., Jointly-Acyclic) if $G(\Sigma)$ (resp., $G_\exists(\Sigma)$) has no cycle going through an $\exists$-arc (resp., is acyclic).

We now recall the notion of marked variable, in order to define the class Sticky (Calì *et al.* 2012b). A variable x of Σ is *marked* if (i) there is $\sigma \in \Sigma$ such that x occurs in $body(\sigma)$ but not in $head(\sigma)$; or (ii) there is $\sigma \in \Sigma$ such that x occurs in $head(\sigma)$ at position π together with some $\sigma' \in \Sigma$ having a marked variable in its body at position π. Accordingly, the *stickiness* condition states that Σ is Sticky if, for each $\sigma \in \Sigma$, x occurs multiple times in $body(\sigma)$ implies x is not marked.

The class Linear (Calì *et al.* 2012a) is based on the *linearity* condition: an ontology Σ belongs to Linear if each rule contains at most one body atoms. This class generalize the class Inclusion-Dependencies (Abiteboul *et al.* 1995; Johnson and Klug 1984) in which rules contain only one body atom and one head atom and the repetition of variables is not allowed neither in the body nor in the head.

The class Guarded (Calì *et al.* 2013) is based on the *guardedness* condition: an ontology Σ belongs to Guarded if for each rule $\sigma \in \Sigma$ there is $\underline{a}$ in $body(\sigma)$ such that $vars_\forall(\sigma) = vars(\underline{a})$. In similar fashion, Σ belongs to Weakly-Guarded if, for each $\sigma \in \Sigma$, there is an atom of $body(\sigma)$ containing all the affected variables of σ.

We recall the *shyness* condition underlying the class Shy (Leone *et al.* 2019). An ontology Σ is Shy if, for each $\sigma \in \Sigma$ the following conditions both hold: (i) if a variable x

occurs in more than one body atom, then x is harmless; (ii) for every pair of distinct dangerous variable z and w in different atoms, $dang(z) \cap dang(w) = \emptyset$.

The class Ward (Gottlob and Pieris 2015) is based on the *wardedness* condition: $\Sigma \in$ Ward if, for each $\sigma \in \Sigma$, there are no dangerous variables in $body(\sigma)$, or there exists an atom $\underline{a} \in body(\sigma)$, called a *ward*, such that (i) all the dangerous variables in $body(\sigma)$ occur in $\underline{a}$, and (ii) each variable of $vars(\underline{a}) \cap vars(body(\sigma) \setminus \{\underline{a}\})$ is harmless.

Having finished with syntactic and semantic conditions, we close the section with a proposition stating their containment relationships (Baget *et al.* 2011; Krötzsch and Rudolph 2011; Leone *et al.* 2019; Baldazzi *et al.* 2022).

Proposition 1
The following classes are pairwise uncomparable, except for:

- Inclusion-Dependencies $\subset$ Joinless, Inclusion-Dependencies $\subset$ Linear;
- Joinless $\subset$ Sticky $\subset$ Sticky-Join $\subset$ FUS;
- Linear $\subset$ Guarded, Linear $\subset$ Protected;
- Guarded $\subset$ Weakly-Guarded, Guarded $\subset$ Fr-Guarded;
- Weakly-Guarded $\subset$ Weakly-Fr-Guarded $\subset$ BTS;
- Fr-Guarded $\subset$ Weakly-Fr-Guarded;
- Datalog $\subset$ Weakly-Guarded, Datalog $\subset$ Protected, Datalog $\subset$ Weakly-Acyclic;
- Protected $\subset$ Ward, Protected $\subset$ Shy $\subset$ SPS;
- Weakly-Acyclic $\subset$ Jointly-Acyclic $\subset$ FES.

Throughout the remainder of the paper, let $\mathbb{E}_{syn}$ denote the set of all 15 decidable syntactic classes reported in Table 1. Analogously, let $\mathbb{E}_{sem}$ denote the set of known decidable abstract classes considered in this paper, namely FES, FUS, BTS, and SPS.

3.3 Autonomous full inclusion dependencies

The aim of this section is to introduce a very simple new class of existential rules called Af-Inds. Additionally, we characterize the main properties of this class.

Definition 3 (*Af-Inds*)
An ontology Σ belongs to Af-Inds (autonomous full inclusion dependencies) if Σ belongs to Inclusion-Dependencies and the following conditions are also satisfied: (1) head predicates do not appear in bodies (autonomous property); (2) rules have no existential variables (full property).

Now, we show that any class $\mathcal{C}$ of TGDs in $\mathbb{E}_{syn} \cup \mathbb{E}_{sem}$ includes the class just defined. Formally, it holds the following.

Proposition 2
Consider a class $\mathcal{C} \in \mathbb{E}_{syn} \cup \mathbb{E}_{sem}$ of TGDs. Then, Af-Inds $\subseteq \mathcal{C}$.

Proof
Thanks to Proposition 1, the statement becomes equivalent to show that (i) Af-Inds $\subseteq$ Inclusion-Dependencies and (ii) Af-Inds $\subseteq$ Datalog. By Definition 3, the class Af-Inds contains all the rules that have only one body and head atom, without repetition of variables

neither in the body nor in the head, and that satisfy the autonomous property (head atom does not appear in bodies) and the full property (rules have only one head atom without existential variables). Accordingly, relation (i) and (ii) are trivially fulfilled. □

We conclude the section by providing the complexity of the class Af-Inds.

Proposition 3
CERT-EVAL[Af-Inds] is in $\mathbf{AC}_0$ in data complexity and $\mathbf{NP}$-complete in combined complexity.

Proof
By Proposition 2, Af-Inds $\subseteq$ Inclusion-Dependencies. Hence, the data complexity of the problem CERT-EVAL[Af-Inds] is inherit from that of CERT-EVAL[Inclusion-Dependencies], that is $\mathbf{AC}_0$. For the combined complexity, we first observe that the problem CERT-EVAL[Af-Inds] is $\mathbf{NP}$-hard, building upon the well-known fact that CERT-EVAL[$\emptyset$] is already $\mathbf{NP}$-hard. The latter refers to the problem of evaluating a query against a database in the absence of an ontology. Secondly, to prove the completeness of the CERT-EVAL[Af-Inds] problem, we show that given a query $q(\mathbf{x})$ and an ontology Σ, it is possible to construct in $\mathbf{NP}$ a CQ $q_\Sigma(\mathbf{x})$ such that $c \in cert(D, \Sigma, q)$ iff $c \in q_\Sigma(D)$, with $\mathbf{c}$ being a tuple in $\mathsf{C}^{|\mathbf{x}|}$. To this aim, for each atom $\underline{a} \in q(\mathbf{x})$, we guess if leave $\underline{a}$ unchanged, or "resolv" $\underline{a}$ with the body of some rule σ in Σ such that $head(\sigma)$ unify with $\underline{a}$. Accordingly, $q_\Sigma(D)$ is polynomial with respect to the input and, finally, it is possible to guess in $\mathbf{NP}$ an homomorphism to check if $c \in q_\Sigma(D)$. □

4 Dyadic pairs of TGDs

In this section we lay the groundwork for the main contribution of the paper, that is the definition of a new decidable class of TGDs called Dyadic-$\mathcal{C}$. To this aim we first introduce some preliminary notions in order to define a *dyadic pair* and, then we conclude with some computational properties.

4.1 Formal definition

We start introducing the concept of *head-ground* set of rules, being roughly "non-recursive" rules in which nulls are neither created nor propagated.

Definition 4 (Head-ground rules)
Consider an ontology Σ. A set $\Sigma' \subseteq \Sigma$ is *head-ground* w.r.t. Σ if the following are true:
(1) $\Sigma' \in$ Datalog; (2) each head atom of Σ' contains only harmless variables w.r.t. Σ;
(3) $h\text{-}preds(\Sigma') \cap b\text{-}preds(\Sigma') = \emptyset$; and (4) $h\text{-}preds(\Sigma') \cap h\text{-}preds(\Sigma \setminus \Sigma') = \emptyset$. ∎

The following example is given to better understand the above definition.

Example 1
Consider the next set of rules:

$$
\begin{aligned}
\sigma_1: \quad & R(x_1, y_1), S(y_1, u_1), T(u_1, v_1) &\rightarrow \quad & \exists\, z_1, w_1\, Q(z_1, w_1) \\
\sigma_2: \quad & C(y_2), R(x_2, z_2) &\rightarrow \quad & S(y_2, z_2) \\
\sigma_3: \quad & D(y_3, z_3), R(x_3, w_3) &\rightarrow \quad & T(x_3, y_3)
\end{aligned}
$$

$$\sigma_4 : \qquad Q(x_4, y_4) \quad \rightarrow \quad \exists\, z_4 A(x_4, z_4)$$
$$\sigma_5 : \quad A(x_5, z_5), D(y_5, z_5) \quad \rightarrow \quad Q(x_5, y_5)$$

A subset of head-ground rule w.r.t. Σ is given by $\Sigma_{\mathrm{HG}} = \{\sigma_2, \sigma_3\}$. In fact, $harmless(\Sigma)$ is the set $\{x_1, y_1, y_2, x_2, z_2, x_3, y_3, z_3, y_5, z_5\}$; hence, according to Definition 4, it is easy to check that (*i*) σ_2 and σ_3 are datalog rules; (*ii*) the head atoms of σ_2 and σ_3 contain only harmless variables; (*iii*) both predicates S and T do not occur in the body of any rule in Σ_{HG}, and (*iv*) both predicates S and T do not occur in the head of any rule in $\{\sigma_1, \sigma_4, \sigma_5\}$. On the contrary, none of the rules in $\{\sigma_1, \sigma_4, \sigma_5\}$ can be part of any head-ground subset of Σ. Indeed, according to Definition 4, both σ_1 and σ_2 violate properties (1) and (2), whereas σ_5 violates property (2). Hence, we observe that the set Σ_{HG} is also maximal. ∎

Having in mind the notion of head-ground set of rules, we can now formally define what is a dyadic pair.

Definition 5 (Dyadic pairs)
Consider a class $\mathcal{C}$ of TGDs. A pair $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ of TGDs is *dyadic* with respect to $\mathcal{C}$ if the next hold: (1) Σ_{HG} is head-ground with respect to $\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}$; and (2) $\Sigma_{\mathcal{C}} \in \mathcal{C}$. ∎

Whenever the above definition applies, we also say, for short, that Π is a $\mathcal{C}$-dyadic pair. Consider the following example to more easily understand the concept of dyadic pair.

Example 2
Consider the following pair $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ of TGDs, where Σ_{HG} is:

$$P(x_1) \quad \rightarrow \quad H_1(x_1)$$
$$P(x_2) \quad \rightarrow \quad H_2(x_2)$$
$$Q(x_3) \quad \rightarrow \quad H_3(x_3).$$

and $\Sigma_{\mathcal{C}}$ is:

$$H_1(x_1) \quad \rightarrow \quad \exists\, y_1, z_1 \; R(y_1, z_1)$$
$$H_2(x_2) \quad \rightarrow \quad \exists\, y_2 \; Q(y_2)$$
$$R(y_3, x_3), H_3(x_3) \quad \rightarrow \quad S(y_3).$$

In particular, Π is a dyadic pair with respect to any $\mathcal{C} \in \{\mathsf{Guarded}, \mathsf{Shy}, \mathsf{Ward.}\}$. To this aim, let $\Sigma = \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}$. It easy computable that $aff(\Sigma) = \{R[1], R[2], Q[1], S[1]\}$, where $aff(R[1]) = \{y_1\}$, $aff(R[2]) = \{z_1\}$, $aff(Q[1]) = \{y_2\}$, and $aff(S[1]) = \{y_1\}$. Accordingly, $harmless(\Sigma) = \{x_1, x_2, x_3\}$, $harmful(\Sigma) = \{y_3\}$ and $dang(\Sigma) = \{y_3\}$. To prove that Π is a dyadic pair, we have first to show that Σ_{HG} is an head-ground set of rules with respect to Σ. Clearly, $\Sigma_{\mathrm{HG}} \in \mathsf{Datalog}$ and each head atom contains only harmless variables; moreover, the head predicates do not appear neither in body atoms of Σ_{HG} nor in head atoms of $\Sigma_{\mathcal{C}}$. Hence, Σ_{HG} is head-ground with respect to Σ. It remains to show that $\Sigma_{\mathcal{C}} \in \mathcal{C}$. We focus on the last rule of $\Sigma_{\mathcal{C}}$, since the first two rules are linear rules, and hence are trivially guarded, shy and ward rules. The last rule belongs to $\mathsf{Guarded}$ since the atom $R(y_3, x_3)$ contains all the universal variables of the rule (guardedness condition); it belong to Shy since the variable x_3 that occurs in two body atoms is harmless (shyness condition); finally, it belongs to Ward since atom $R(y_3, x_3)$ is the ward that contains the dangerous variables (y_3) and shares with the rest of the body only harmless variables (x_3) (wardedness condition). ∎

The next step is to extend the query answering problem – classically defined over an ontology – over a dyadic pair. Therefore, we extend both notions of chase and certain answers for a dyadic pair. Accordingly, given a dyadic pair $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$, we define

$$dp\text{-}chase(D, \Pi) = chase(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \tag{3}$$

and

$$dp\text{-}cert(q, D, \Pi) = cert(q, D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}). \tag{4}$$

Now we can fix the problem studied in the rest of the paper.

PROBLEM: DP-CERT-EVAL$[\mathcal{C}]$.
    INPUT: A database D, a $\mathcal{C}$-dyadic pair $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ of TGDs, a CQ $q(\mathbf{x})$, and a tuple $\mathbf{c} \in \mathsf{C}^{|\mathbf{x}|}$.
    QUESTION: Does $\mathbf{c} \in dp\text{-}cert(q, D, \Pi)$ holds?

4.2 Computational properties

For the rest of the section, fix a decidable class $\mathcal{C}$ of TGDs. Given a database D and a $\mathcal{C}$-dyadic pair Π of TGDs, we define the following set of ground atoms:

$$gra(D, \Pi) = \{\underline{a} \in chase(D, \Pi) \mid \Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}}) \wedge preds(\underline{a}) \in h\text{-}preds(\Sigma_{\mathrm{HG}})\}. \tag{5}$$

Our idea is to reduce query answering over a dyadic pair Π to query answering over $\mathcal{C}$, the latter being decidable by assumption.

Theorem 1
Consider a database D, a $\mathcal{C}$-dyadic pair $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ of TGDs, and a conjunctive query $q(\mathbf{x})$. Let $D^+ = D \cup gra(D, \Pi)$. It holds that $dp\text{-}cert(q, D, \Pi) = cert(q, D^+, \Sigma_{\mathcal{C}})$.

Proof
Consider $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$. By equation (4), $dp\text{-}cert(q, D, \Pi) = cert(q, D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})$. Moreover, by fixing any arbitrary $|\mathbf{x}|$-ary tuple $\mathbf{c}$ of constants, it holds that $\mathbf{c} \in cert(q, D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \Leftrightarrow D \cup \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}} \models q(\mathbf{c})$ and $\mathbf{c} \in cert(q, D^+, \Sigma_{\mathcal{C}}) \Leftrightarrow D^+ \cup \Sigma_{\mathcal{C}} \models q(\mathbf{c})$. Let $q' = q(\mathbf{c})$. Accordingly, the thesis boils down to showing that

$$D \cup \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}} \models q' \Leftrightarrow D^+ \cup \Sigma_{\mathcal{C}} \models q'.$$

[$\Rightarrow$] Assume that $D \cup \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}} \models q'$ holds. Hence, $chase(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \models q'$. Given that $gra(D, \Pi) \subseteq chase(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})$, it holds that $chase(D \cup gra(D, \Pi), \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \models q'$. Moreover, since $gra(D, \Pi)$ contains all the auxiliary ground consequences of Σ_{HG}, the latter becomes equivalent to $chase(D \cup gra(D, \Pi), \Sigma_{\mathcal{C}}) \models q'$. Hence, $D \cup gra(D, \Pi) \cup \Sigma_{\mathcal{C}} \models q'$, that is $D^+ \cup \Sigma_{\mathcal{C}} \models q'$.

[$\Leftarrow$] Assume that $D^+ \cup \Sigma_{\mathcal{C}} \models q'$, hence $chase(D^+, \Sigma_{\mathcal{C}}) \models q'$. Since $\Sigma_{\mathcal{C}} \subseteq \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}$, it holds that $chase(D^+, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \models q'$. By hypothesis, $gra(D, \Pi) \subseteq chase(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})$; hence $chase(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \models q'$, that is $D \cup \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}} \models q'$. □

According to Theorem 1, to solve DP-CERT-EVAL$[\mathcal{C}]$ one can first "complete" D and then performing classical query evaluation. To this aim we design Algorithms 1 and 2. The

---

Algorithm 1: $\mathsf{DpCertEval}_{[\mathcal{C}]}(q, D, \Pi, \mathbf{c})$

Input: A database D, a dyadic pair Π, a CQ $q(\mathbf{x})$, and a tuple $\mathbf{c} \in \mathsf{C}^{|\mathbf{x}|}$

1 $D^+ := \mathsf{Complete}_{[\mathcal{C}]}(D, \Pi)$;

2 **return** $\mathbf{c} \in cert(q, D^+, \Sigma_{\mathcal{C}})$;

---

correctness of the latter will be proved in Proposition 4. Consequently, the correctness of the former is guaranteed by Theorem 1.

In particular, given a database D and a dyadic pair $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$, Algorithm 2 iteratively constructs the set $D^+ = D \cup gra(D, \Pi)$, with $gra(D, \Pi)$ being the set defined by equation (5). Roughly speaking, the first two instructions are required, respectively, to add D to D^+ and to initialize a temporary set $\tilde{D}$ used to store ground consequences derived from Σ_{HG}. The rest of the algorithm is an iterative procedure that computes the certain answers (instruction 5) to the queries constructed from the rules of Σ_{HG} (instruction 4) and completes the initial database D (instruction 7) until no more auxiliary ground atoms can be produced (instruction 6). We point out that, in general, $\tilde{D} \subseteq gra(D, \Pi)$ holds; in particular, $\tilde{D} = gra(D, \Pi)$ holds in the last execution of instruction 7 or, equivalently, when the condition $D \cup \tilde{D} \supset D^+$ examined at instruction 6 is false, since all the auxiliary ground atoms have been added to D^+.

Before we prove that Algorithm 2 always terminates and correctly constructs D^+, we show the following preliminary lemma.

Algorithm 2: $\mathsf{Complete}_{[\mathcal{C}]}(D, \Pi)$

Input: A database D and a $\mathcal{C}$-dyadic pair $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$

Output: The set D^+ of ground atoms

1 $D^+ := D$

2 $\tilde{D} := \emptyset$

3 **for** each rule of the form $\Phi(\mathbf{x}, \mathbf{y}) \to \mathrm{H}_i(\mathbf{x})$ in Σ_{HG} **do**

4 $\quad q := \langle \mathbf{x} \rangle \leftarrow \Phi(\mathbf{x}, \mathbf{y})$

5 $\quad \tilde{D} = \tilde{D} \cup \{\mathrm{H}_i(\mathbf{t}) \mid \mathbf{t} \in cert(q, D^+, \Sigma_{\mathcal{C}})\}$

6 **if** $(D \cup \tilde{D} \supset D^+)$ **then**

7 $\quad D^+ := D \cup \tilde{D}$

8 $\quad$ **go to** instruction 2

9 **return** D^+

Lemma 1

Consider a database D and a set Σ of TGDs. Let $\Sigma' \subseteq \Sigma$ and $X \subseteq chase^{\perp}(D, \Sigma) \setminus D$. Then, $chase(D \cup X, \Sigma') \subseteq chase(D, \Sigma)$.

Proof

Let $X = \{\underline{a}_1, ..., \underline{a}_n\}$. For each $j \in [n]$, let $lev(\underline{a}_i)$ be the level of $\underline{a}_i$ inside $chase(D, \Sigma)$. Let $p = \max_{j \in [n]}\{lev(\underline{a}_i)\}$. The proof proceeds by induction on the level i of $chase(D \cup X, \Sigma')$.

Base case: $i = 1$. We want to prove that $chase^1(D \cup X, \Sigma') \subseteq chase(D, \Sigma)$. Let $\underline{a}$ be an atom of $chase^1(D \cup X, \Sigma')$ generated exactly at level $i = 1$. By definition, $\underline{a}$ is obtained

due to some trigger $\langle \sigma, h \rangle$ such that h maps $body(\sigma)$ to $D \cup X$. If $\underline{a} \in chase^p(D, \Sigma)$, then the claim holds trivially. Otherwise, we can show that $\underline{a} \in chase^{p+1}(D, \Sigma)$. Indeed, since h maps $body(\sigma)$ to $D \cup X$, then h is also a trigger involved at level $p + 1$ since it maps $body(\sigma)$ to $chase^p(D, \Sigma)$. In particular, h maps at least one atom of $body(\sigma)$ to some atom $\underline{a}_k \in X$ such that $lev(\underline{a}_k) = p$. Since, by definition, nulls introduced during the chase functionally depend on the involved triggers, then $\underline{a}$ necessarily belongs to $chase^{p+1}(D, \Sigma)$.

Induction step: $i = \ell$. Given that for every level $i \leq \ell - 1$, $chase^i(D \cup X, \Sigma') \subseteq chase(D, \Sigma)$ (*induction hypothesis*), we prove that $chase^\ell(D \cup X, \Sigma') \subseteq chase(D, \Sigma)$ holds, too. Let β be an atom of $chase^\ell(D \cup X, \Sigma')$ generated exactly at level $i = \ell$. By definition, β is obtained via some trigger $\langle \sigma', h' \rangle$ such that h' maps $body(\sigma')$ to atoms with level at most $\ell - 1$. Accordingly, by induction hypothesis, h' maps $body(\sigma')$ also to $chase(D, \Sigma)$. Hence, since the processing order of rules and triggers does not change the result of the chase and nulls functionally depend on the involved triggers, it follow that also $\beta \in chase(D, \Sigma)$. $\square$

With the next proposition, we prove that Algorithm 2 always terminates and correctly constructs D^+.

Proposition 4

Consider a database D and a $\mathcal{C}$-dyadic pair Π of TGDs. It holds that Algorithm 2 both terminates and computes $D^+ = D \cup gra(D, \Pi)$.

Proof

Let $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$. We proceed by proving first the termination of Algorithm 2 and then its correctness.

Termination. To prove the termination of Algorithm 2, it suffices to show that each instruction alone always terminates and that the overall procedure never falls into an infinite loop. First, observe that $|gra(D, \Pi)| \leq |h\text{-}preds(\Sigma_{\mathrm{HG}})| \cdot d^\mu$, where $d = |consts(D)|$ and $\mu = \max_{P \in h\text{-}preds(\Sigma_{\mathrm{HG}})} arity(P)$. Instructions 1, 2, 4, 8, and 9 trivially terminate. Instructions 6 and 7 both terminate, since $\tilde{D} \subseteq gra(D, \Pi)$ always holds (see correctness below). Each time instruction 3 is reached, the **for**-loop simply scans the set Σ_{HG}, which is finite by definition. Concerning instruction 5, it suffices to observe that its termination relies on the termination of CERT-EVAL$[\mathcal{C}]$ – which is true by hypothesis – and on the fact that, for each query q, to construct the set $\{\mathrm{H}_i(\mathbf{t}) \mid \mathbf{t} \in cert(q, D^+, \Sigma_{\mathcal{C}})\}$, the problem CERT-EVAL$[\mathcal{C}]$ must be solved at most d^μ times, where d^μ is the maximum number of tuples $\mathbf{t}$ for which the check $\mathbf{t} \in cert(q, D^+, \Sigma_{\mathcal{C}})$ has to be performed. Since each instruction alone terminates, it remains to analyze the overall procedure. It contains two loops. The first, namely the **for**-loop at instruction 3, is not problematic; indeed, we shown that it locally terminates. The second one, namely the **go to**-loop, depends on the evaluation of the **if**-instruction, which can be executed at most $|gra(D, \Pi)|$ times. Thus, also the **go to**-loop does the same.

Correctness. We now claim that Algorithm 2 correctly completes the database. Let D^+ be the output of Algorithm 2. Our claim is that $D^+ = D \cup gra(D, \Pi)$.

Inclusion 1 ($D^+ \supseteq D \cup gra(D,\Pi)$). Assume, by contradiction, that $D \cup gra(D,\Pi)$ contains some atom that does not belong to D^+. This means that there exists some $j > 0$ such that both $\bar{D} = ((D \cup gra(D,\Pi)) \cap chase^{j-1}(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})) \subseteq D^+$ and $((D \cup gra(D,\Pi)) \cap chase^j(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})) \setminus D^+ \neq \emptyset$ hold. Thus, there exists some $\underline{a} \in chase^j(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})$ whose level is exactly j and that does not belong to D^+. Let $\langle \sigma, h \rangle$ be the trigger used by the chase to generate $\underline{a}$, where σ is of the form $\Phi(\mathbf{x}, \mathbf{y}) \rightarrow \mathrm{H}(\mathbf{x})$. Clearly, h maps $\Phi(\mathbf{x}, \mathbf{y})$ to $chase^{j-1}(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})$, and we also have that $\underline{a} = \mathrm{H}(h(\mathbf{x}))$. Consider now the query $q = \langle \mathbf{x} \rangle \leftarrow \Phi(\mathbf{x}, \mathbf{y})$ constructed from σ by Algorithm 2 at instruction 4. Thus, $chase^{j-1}(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \models q(h(\mathbf{x}))$ holds. Since $\bar{D} \subseteq D^+$, we have that $chase^{j-1}(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \subseteq chase(\bar{D}, \Sigma_{\mathcal{C}}) \subseteq chase(D^+, \Sigma_{\mathcal{C}})$. Hence, $chase(D^+, \Sigma_{\mathcal{C}}) \models q(h(\mathbf{x}))$, namely $h(\mathbf{x}) \in cert(q, D^+, \Sigma_{\mathcal{C}})$ and, thus, $\underline{a} \in D^+$, which is a contradiction.

Inclusion 2 ($D^+ \subseteq D \cup gra(D,\Pi)$). Let D^+ be the set produced by Algorithm 2. Let ℓ be the number of time instruction 7 of Algorithm 2 is executed. At each execution $i \in [\ell]$ of instruction 7, the algorithm computes the set $\tilde{D}_i$ containing only auxiliary ground atoms, and produces the set $D_i^+ = D \cup \tilde{D}_i$. By construction, $D^+ = D \cup \tilde{D}_\ell$. Let $\tilde{D}_0 = \emptyset$, $D_0^+ = D \cup \tilde{D}_0$, and $I_i = \tilde{D}_i \setminus \tilde{D}_{i-1}$, for each $i \in [\ell]$. We show that $D \cup \tilde{D}_\ell \subseteq D \cup gra(D,\Pi)$, that is $\tilde{D}_\ell \subseteq gra(D,\Pi)$. We proceed by induction on the number ℓ of iterations.

Base case: Let $i = 1$. We claim that $\tilde{D}_1 \subseteq gra(D,\Pi)$. By construction, the set $\tilde{D}_1 = \{\mathrm{H}_i(\mathbf{t}) \mid i \in [|\Sigma_{\mathrm{HG}}|] \wedge \mathbf{t} \in cert(q, D_0^+, \Sigma_{\mathcal{C}})\}$. Since $D_0^+ = D$ and the component $\Sigma_{\mathcal{C}}$ does not produce any atom in the first iteration of the algorithm, the latter is equal to $\{\mathrm{H}(\mathbf{t}) \mid i \in [|\Sigma_{\mathrm{HG}}|] \wedge \mathbf{t} \in cert(q, D, \emptyset)\} = \{\underline{a} \in chase^1(D, \Sigma_{\mathrm{HG}}) \mid preds(\underline{a}) \in h\text{-}preds(\Sigma_{\mathrm{HG}})\} \subseteq \{\underline{a} \in chase(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \mid preds(\underline{a}) \in h\text{-}preds(\Sigma_{\mathrm{HG}})\} = gra(D,\Pi)$.

Induction step: Given that, for $i = \ell - 1$, $\tilde{D}_{\ell-1} \subseteq gra(D,\Pi)$ (*induction hypothesis*), we prove that $\tilde{D}_\ell \subseteq gra(D,\Pi)$ holds, too. By construction $\tilde{D}_\ell = \{\mathrm{H}_i(\mathbf{t}) \mid i \in [|\Sigma_{\mathrm{HG}}|] \wedge \mathbf{t} \in cert(q, D_{\ell-1}^+, \Sigma_{\mathcal{C}})\} = \{\mathrm{H}_i(\mathbf{t}) \mid i \in [|\Sigma_{\mathrm{HG}}|] \wedge \mathbf{t} \in cert(q, D \cup \tilde{D}_{\ell-1}, \Sigma_{\mathcal{C}})\} = \{\underline{a} \in chase(D \cup \tilde{D}_{\ell-1}, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \mid preds(\underline{a}) \in h\text{-}preds(\Sigma_{\mathrm{HG}})\}$. Since the set $D \cup \tilde{D}_{\ell-1}$ already contains all the ground consequences of Σ_{HG}, the latter is equivalent to $\{\underline{a} \in chase(D \cup \tilde{D}_{\ell-1}, \Sigma_{\mathcal{C}}) \mid preds(\underline{a}) \in h\text{-}preds(\Sigma_{\mathrm{HG}})\}$. Applying Lemma 1 together with the induction hypothesis, the last one is a subset of $\{\underline{a} \in chase(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}) \mid preds(\underline{a}) \in h\text{-}preds(\Sigma_{\mathrm{HG}})\} = gra(D,\Pi)$. □

We conclude the section by proving the decidability of the problem DP-CERT-EVAL[$\mathcal{C}$], under the assumption that CERT-EVAL[$\mathcal{C}$] is decidable.

Theorem 2
If CERT-EVAL[$\mathcal{C}$] is decidable, then DP-CERT-EVAL[$\mathcal{C}$] is decidable.

Proof
To prove the decidability of DP-CERT-EVAL[$\mathcal{C}$] we design Algorithm 1. Let D be a database, $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ a dyadic pair, $q(\mathbf{x})$ a CQ, and $\mathbf{c}$ a tuple in C^{x}. Clearly, step 1 always terminates since it recalls Algorithm 2 that, as shown in Proposition 4, always terminates and correctly constructs the set D^+. By Theorem 1, checking if $\mathbf{c} \in dp\text{-}cert(q, D, \Pi)$ boils down to checking if $\mathbf{c} \in cert(q, D^+, \Sigma_{\mathcal{C}})$, that is decidable by hypothesis. Hence, step 2 never falls in a loop and Algorithm 1 correctly computes DP-CERT-EVAL[$\mathcal{C}$]. □

5 Dyadic decomposable sets

In this section we introduce a novel general condition that allows to define, from any decidable class $\mathcal{C}$ of ontologies, a new decidable class called Dyadic-$\mathcal{C}$ enjoying desirable properties. The union of all the Dyadic-$\mathcal{C}$ classes, with $\mathcal{C}$ being any decidable class of TGDs, forms what we call *dyadic decomposable sets*, which encompass and generalize any other existing decidable class, including those based on semantic conditions.

We start the section by providing a classification of atoms of the rule-body, according to where dangerous variables appear; then we define the class Dyadic-$\mathcal{C}$ proving that the query answer problem over this class is decidable.

Definition 6 (Atoms classification)
Consider a set Σ of TGDs and a rule $\sigma \in \Sigma$. An atom $\underline{a}$ of $body(\sigma)$ is σ-*problematic* if (*i*) $\underline{a}$ contains a dangerous variable w.r.t. Σ, or (*ii*) $\underline{a}$ is connected to some σ-*problematic* atom via some harmful variable. The set of all the *problematic* atoms of σ is denoted by $p\text{-}atoms(\sigma)$, whereas $s\text{-}atoms(\sigma) = body(\sigma) \setminus p\text{-}atoms(\sigma)$ denotes the set of all the *safe* atoms of σ.   ∎

We highlight that $p\text{-}atoms$ and $s\text{-}atoms$ can share only harmless variables. The next example is to clarify the above definition.

Example 3
Consider the database $D = \{L(a), R(a, b)\}$, and the following set Σ of TGDs:

$$
\begin{array}{rlrl}
\sigma_1 : & & L(x_1) & \rightarrow \ \exists\, y_1 \ P(y_1, x_1) \\
\sigma_2 : & & P(x_2, y_2) & \rightarrow \ \exists\, z_2 \ Q(y_2, z_2, x_2) \\
\sigma_3 : & & P(z_3, x_3), Q(x_3, y_3, z_3) & \rightarrow \ S(z_3) \\
\sigma_4 : & P(x_4, y_4), Q(y_4, z_4, w_4), S(w_4), R(u_4, v_4) & \rightarrow \ T(x_4, z_4, u_4).
\end{array}
$$

We focus on rule σ_4. Taking into account that $aff(\Sigma) = \{P[1], S[1], Q[2], Q[3]\}$, it follows that $dang(\Sigma) = \{x_2, x_4, z_3, z_4\}$, $harmful(\Sigma) = \{x_2, x_4, y_3, z_3, z_4, w_4\}$, and $harmless(\Sigma) = \{x_1, x_3, y_2, y_4, u_4, v_4\}$. Hence, $p\text{-}atoms(\sigma_4) = \{P(x_4, y_4), Q(y_4, z_4, w_4), S(w_4)\}$, whereas $s\text{-}atoms(\sigma_4) = \{R(u_4, v_4)\}$.   ∎

The second step consists in selecting variables shared by $p\text{-}atoms$ and $s\text{-}atoms$ together with the harmless frontier variables appearing in $s\text{-}atoms$. The latter can be expressed via the set $hf(\sigma) = \{x \in vars(s\text{-}atoms(\sigma)) \mid x \in harmless(\sigma) \wedge x \in \mathsf{vars}_\frown(\sigma)\}$. Then, we define $vars_\star(\sigma) = \{vars(p\text{-}atoms(\sigma)) \cap vars(s\text{-}atoms(\sigma))\} \cup hf(\sigma)$. Trivially, it holds that $vars_\star(\sigma) \subseteq vars(s\text{-}atoms(\sigma))$. Finally, let

$$hg(\sigma) : s\text{-}atoms(\sigma) \rightarrow \mathtt{Aux}_\sigma(vars_\star(\sigma)),$$

$$main(\sigma) : \mathtt{Aux}_\sigma(vars_\star(\sigma)), \ p\text{-}atoms(\sigma) \rightarrow \exists\, vars_\exists(\sigma) \ head(\sigma),$$

$$hg(\Sigma) = \bigcup_{\sigma \in \Sigma} hg(\sigma) \qquad \text{and} \qquad main(\Sigma) = \bigcup_{\sigma \in \Sigma} main(\sigma).$$

By considering again Example 3, we have that $vars_\star(\sigma_4) = \{u_4, y_4\}$ and also that $main(\sigma_4) : \mathtt{Aux}_{\sigma_4}(u_4, y_4), P(x_4, y_4), Q(y_4, z_4, w_4), S(w_4, y_4) \rightarrow T(x_4, z_4, u_4)$.

In the special case in which a variable x of $vars_\star(\sigma)$ occurs $n > 1$ times in $head(\sigma)$, then x also occurs n times in the head of $hg(\sigma)$. Accordingly, x occurs with different

names (e.g., $x_1, ..., x_n$) both in the head and in the body of $main(\sigma)$. For example, if the ontology contains only the rule $\sigma : P(x) \rightarrow R(x, x)$, then $hg(\sigma) : P(x) \rightarrow \mathtt{Aux}_\sigma(x, x)$ and $main(\sigma) : \mathtt{Aux}_\sigma(x_1, x_2) \rightarrow R(x_1, x_2)$. Clearly, the latter two rules together are equivalent to σ w.r.t. to the schema $\{P, R\}$. We prefer to keep the formal definition of $hg(\Sigma)$ and $main(\Sigma)$ light without formalizing such special cases.

We are now ready to formally introduce the class Dyadic-$\mathcal{C}$.

Definition 7 (*Dyadic-$\mathcal{C}$*)
Consider a class $\mathcal{C}$ of TGDs such that CERT-EVAL[$\mathcal{C}$] is decidable. We say that Σ belongs to Dyadic-$\mathcal{C}$ if Σ belongs to $\mathcal{C}$ or if $main(\Sigma)$ belongs to $\mathcal{C}$. ■

According to the previous definition, one can easily state the following property.

Proposition 5
Consider a class $\mathcal{C}$ of TGDs. It holds that $\mathcal{C} \subseteq$ Dyadic-$\mathcal{C}$.

By Definition 7, to check if an ontology Σ belongs to Dyadic-$\mathcal{C}$, one has to verify if $\Sigma \in \mathcal{C}$, or $main(\Sigma) \in \mathcal{C}$. We observe that the construction of the set $main(\Sigma)$ explained above, is polynomial (indeed linear) with respect to the size of Σ. Hence, the following result holds.

Theorem 3
Consider a class $\mathcal{C}$ of TGDs and assume that checking whether an ontology belongs to $\mathcal{C}$ is doable in some complexity class $\mathbb{C} \supseteq \mathbf{PTIME}$. Then, checking whether an ontology belongs to Dyadic-$\mathcal{C}$ is decidable and it belongs to $\mathbb{C}$.

Proof
We start recalling that, by Definition 7, an ontology Σ belongs to Dyadic-$\mathcal{C}$ if (i) $\Sigma \in \mathcal{C}$, or (ii) $main(\Sigma) \in \mathcal{C}$. Accordingly, checking condition (i) is doable in some complexity class $\mathbb{C} \supseteq \mathbf{PTIME}$, by assumption; otherwise, the construction of the set $main(\Sigma)$ is done by a procedure that is polynomial (indeed linear) with respect to the size of Σ and, hence, always terminates. Accordingly, checking condition (ii) is also decidable and doable in the complexity class $\mathbb{C}$. □

The next step is to prove the existence of a $\mathcal{C}$-dyadic pair for any ontology $\Sigma \in$ Dyadic-$\mathcal{C}$.

Theorem 4
Consider a set $\Sigma \in$ Dyadic-$\mathcal{C}$. There exists a $\mathcal{C}$-dyadic pair $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_\mathcal{C})$ of TGDs such that $\Sigma_{\mathrm{HG}} \cup \Sigma_\mathcal{C} \equiv_{sch(\mathbf{\Sigma})} \Sigma$. In particular,
  1. If $\Sigma \in \mathcal{C}$, then $\Sigma_{\mathrm{HG}} = \emptyset$ and $\Sigma_\mathcal{C} = \Sigma$;
  2. If $\Sigma \notin \mathcal{C}$, then $\Sigma_{\mathrm{HG}} = hg(\Sigma)$ and $\Sigma_\mathcal{C} = main(\Sigma)$.

Proof
We claim that for each ontology $\Sigma \in$ Dyadic-$\mathcal{C}$ it is possible to construct a $\mathcal{C}$-dyadic pair $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_\mathcal{C})$ of TGDs such that $\Sigma_{\mathrm{HG}} \cup \Sigma_\mathcal{C} \equiv_{sch(\mathbf{\Sigma})} \Sigma$.

According to Definition 5, we recall that a pair $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_\mathcal{C})$ is $\mathcal{C}$-dyadic if (i) Σ_{HG} is head-ground with respect to $\Sigma_{\mathrm{HG}} \cup \Sigma_\mathcal{C}$ and (ii) $\Sigma_\mathcal{C} \in \mathcal{C}$. Assume first that $\Sigma \in \mathcal{C}$. Then, trivially the pair $(\emptyset, \Sigma)$ is a $\mathcal{C}$-dyadic pair. Assume now that $\Sigma \notin \mathcal{C}$ and let $\Pi = (hg(\Sigma), main(\Sigma))$. Property (ii) is satisfied since by hypothesis $\Sigma \in$ Dyadic-$\mathcal{C}$; hence, it follows by definition that $main(\Sigma) \in \mathcal{C}$. It remains to show property (i). According to Definition 4, the set $hg(\Sigma)$ has to satisfy four properties. Property 1 and 2 are trivially

---

Algorithm 3: $\mathsf{CertEval}_{[\mathsf{Dyadic}\text{-}\mathcal{C}]}(q, D, \Sigma, \mathbf{c})$

Input: A database D, a $\Sigma \in \mathsf{Dyadic}\text{-}\mathcal{C}$, a CQ $q(\mathbf{x})$, and a tuple $\mathbf{c} \in \mathsf{C}^{|\mathbf{x}|}$

1 $\Sigma_{\mathrm{HG}} = hg(\Sigma)$;
2 $\Sigma_{\mathcal{C}} = main(\Sigma)$;
3 $\Pi = (\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$;
4 return $\mathsf{DpCertEval}_{[\mathcal{C}]}(q, D, \Pi, \mathbf{c})$;

---

fulfilled since, by construction, for each $\sigma \in \Sigma$, $hg(\sigma)$ is a datalog rule and each head atom contains only harmless variables with respect to $hg(\Sigma) \cup main(\Sigma)$. Property 3 and 4 state that $h\text{-}preds(hg(\Sigma)) \cap b\text{-}preds(hg(\Sigma)) = \emptyset$ and $h\text{-}preds(hg(\Sigma)) \cap h\text{-}preds(main(\Sigma)) = \emptyset$. These hold since, by construction, $h\text{-}preds(hg(\Sigma)) = \{\mathtt{Aux}_\sigma : \sigma \in \Sigma\}$, where each $\mathtt{Aux}_\sigma$ is a predicate that does not occur neither in any body of $hg(\Sigma)$ nor in any head of $main(\Sigma)$.

Concerning the equivalence between $\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}$ and Σ, we can observe that it easily comes from the shape of $hg(\sigma)$ and $main(\sigma)$ with respect to each original rule $\sigma \in \Sigma$. Indeed, first, the body of σ is first partitioned in $s\text{-}atoms(\sigma)$ and $p\text{-}atoms(\sigma)$. Second, all the atoms if $s\text{-}atoms(\sigma)$ form the body of $hg(\sigma)$. Then, all the variables of $hg(\sigma)$ that are in join with $p\text{-}atoms(\sigma)$ or are in the head of σ are collected in $\mathtt{Aux}_\sigma(vars_\star(\sigma))$. Finally, $\mathtt{Aux}_\sigma(vars_\star(\sigma))$ is put in conjunction with $p\text{-}atoms(\sigma)$ to form the body of $main(\sigma)$. Such a way of decomposing a rule σ is well-known to be correct for query answering purposes even when the variables in the auxiliary atom are harmful. $\square$

It remains to show that $\mathsf{Dyadic}\text{-}\mathcal{C}$ is decidable. We rely on Algorithm 3 together with Theorem 1 and Proposition 4 to state the following result.

Theorem 5
Consider a decidable class $\mathcal{C}$ of TGDs. Then, CERT-EVAL[$\mathsf{Dyadic}\text{-}\mathcal{C}$] is decidable.

Proof
To prove the statement we provide the terminating Algorithm 3. Let $\Sigma \in \mathsf{Dyadic}\text{-}\mathcal{C}$ an ontology. Instructions 1 and 2 of the algorithm are introduced in order to construct the components $(hg(\Sigma), main(\Sigma))$ of a dyadic pair Π, which is successively initialized at instruction 3. Of course, the construction of Π is based on a polynomial procedure with respect to the size of the input Σ, hence these instructions always terminates. Finally, instruction 4 returns the result of the evaluation of the problem DP-CERT-EVAL[$\mathcal{C}$]. To solve the latter, is invoked Algorithm 1, which in turn invokes Algorithm 2; their correctness is guaranteed by Theorem 1 and Proposition 4, respectively. Accordingly, CERT-EVAL[$\mathsf{Dyadic}\text{-}\mathcal{C}$] is decidable. $\square$

Finally, we conclude the section by proving that, for any class $\mathcal{C}$ including the class $\mathsf{Af}\text{-}\mathsf{Inds}$ defined in Section 3.3, the class $\mathsf{Dyadic}\text{-}\mathcal{C}$ includes $\mathsf{Datalog}$.

Theorem 6
Consider a class $\mathcal{C}$ of TGDs. If $\mathcal{C} \supseteq \mathsf{Af}\text{-}\mathsf{Inds}$, then $\mathsf{Datalog} \subseteq \mathsf{Dyadic}\text{-}\mathcal{C}$.

Proof
Consider a datalog rule $\sigma : \Phi(\mathbf{x}, \mathbf{y}) \to P(\mathbf{x})$. Then, it can be decomposed into the following two rules: $hg(\sigma) : \Phi(\mathbf{x}, \mathbf{y}) \to \mathtt{Aux}_\sigma(\mathbf{x})$ and $main(\sigma) : \mathtt{Aux}_\sigma(\mathbf{x}) \to P(\mathbf{x})$. Trivially,

$hg(\sigma) \in \Sigma_{\mathrm{HG}}$; it remains to show that $main(\sigma) \in \mathsf{Af\text{-}Inds}$. According to Definition 3, the full property immediately follows since datalog rules do no have existential variables, whereas the autonomous property holds since $body(main(\sigma))$ contains only the fresh predicate H. In particular, $main(\sigma) \in \mathcal{C}$, since by assumption $\mathcal{C} \supseteq \mathsf{Af\text{-}Inds}$; hence, the thesis follows. □

6 Computational complexity of query answering

In this section we study the complexity of the CERT-EVAL problem over dyadic existential rules. We start analyzing the data complexity of the problem and then the combined complexity.

Theorem 7
Consider a class $\mathcal{C}$ of TGDs. In data complexity, if CERT-EVAL$[\mathcal{C}]$ belongs to some decidable complexity class $\mathbb{C}$, then the following hold:

1. If $\mathbb{C} \subseteq \mathbf{PTIME}$, then CERT-EVAL$[\mathsf{Dyadic\text{-}}\mathcal{C}]$ is in $\mathbf{PTIME}$;
2. If $\mathbb{C} \supseteq \mathbf{PTIME}$, then CERT-EVAL$[\mathsf{Dyadic\text{-}}\mathcal{C}]$ is in $\mathbf{PTIME}^{\mathbb{C}}$;
3. If $\mathbb{C} \supseteq \mathbf{PTIME}$ is deterministic[2] and CERT-EVAL$[\mathcal{C}]$ is $\mathbb{C}$-complete, then it holds that CERT-EVAL$[\mathsf{Dyadic\text{-}}\mathcal{C}]$ is $\mathbb{C}$-complete too;
4. If $\mathcal{C} \supseteq \mathsf{Af\text{-}Inds}$, then CERT-EVAL$[\mathsf{Dyadic\text{-}}\mathcal{C}]$ is $\mathbf{PTIME}$-hard.

Proof
To prove the theorem, we rely on the complexity of Algorithm 3. Let D be a database, $\Sigma \in \mathsf{Dyadic\text{-}}\mathcal{C}$ an ontology, $q(\mathbf{x})$ a CQ, and $\mathbf{c} \in \mathsf{C}^{|\mathbf{x}|}$ a tuple. Moreover, let $d = |consts(D)|$ and $\mu = \max_{P \in h\text{-}preds(\Sigma_{\mathrm{HG}})} arity(P)$. As shown in Theorem 5, Algorithm 3 terminates and correctly decides CERT-EVAL$[\mathsf{Dyadic\text{-}}\mathcal{C}]$. More specifically, instructions 1 and 2 are introduced in order to construct, respectively, the first component $hg(\Sigma)$ and the second component $main(\Sigma)$ of a dyadic pair. The procedure used to build these sets is polynomial (indeed linear) with respect to the size of the ontology Σ in input. By neglecting the "trivial" instruction 3, the computational cost of the Algorithm mainly depends on instruction 4, that is on the invocation of Algorithm 1, which in turn invokes Algorithm 2 in order to compute the completed database D^+. As stated in Proposition 4, Algorithm 2 always terminates. In particular, the size of the set D^+ is at most $|h\text{-}preds(\Sigma_{\mathrm{HG}})| \cdot d^\mu$; the problem CERT-EVAL$[\mathcal{C}]$, at instruction 5, is called d^μ times, and the **for**-loop at instruction 3 is executed $|\Sigma_{\mathrm{HG}}|$ times. Therefore, by ignoring the computational costs of the oracle (i.e., checking whether $\mathbf{t} \in cert(q, D^+, \Sigma_\mathcal{C})$), Algorithm 2 overall performs a number of step that is linear in $|\Sigma_{\mathrm{HG}}| \cdot |h\text{-}preds(\Sigma_{\mathrm{HG}})| \cdot d^{2\mu}$. Indeed, this value is also an upper bound for the number of calls to the oracle. Since we are in data complexity, the following parameters are bounded: the maximum arity μ, the size of the sets Σ_{HG} and $\Sigma_\mathcal{C}$, as well as the size and the number of each query q constructed at instruction 4 of Algorithm 2. Hence, the latter calls polynomially many times the problem CERT-EVAL$[\mathcal{C}]$. Accordingly, Algorithm 3 is polynomial and in turn it invokes polynomially many times an oracle to compute CERT-EVAL$[\mathcal{C}]$. Hence, if $\mathbb{C} \subseteq \mathbf{PTIME}$, trivially CERT-EVAL$[\mathsf{Dyadic\text{-}}\mathcal{C}] \in \mathbf{PTIME}$;

---

[2] By "deterministic" we mean that CERT-EVAL$[\mathsf{Dyadic\text{-}}\mathcal{C}]$ can be solved by a deterministic Turing machine. We refer to the classes $\mathbf{PTIME}, \mathbf{PSPACE}, \mathbf{i\text{-}EXPTIME}, \mathbf{i\text{-}EXPSPACE}$, for all $i \geq 1$.

Table 2. *Data complexity comparison of* CERT-EVAL$[\mathcal{C}]$ *with* CERT-EVAL$[\textit{Dyadic-}\mathcal{C}]$

Class $\mathcal{C}$	$\mathcal{C}$	Dyadic-$\mathcal{C}$
Weakly-(Fr)-Guarded	**EXPTIME**-complete	**EXPTIME**-complete
(Fr)-Guarded	**PTIME**-complete	**PTIME**-complete
Weakly-Acyclic	**PTIME**-complete	**PTIME**-complete
Jointly-Acyclic	**PTIME**-complete	**PTIME**-complete
Datalog	**PTIME**-complete	**PTIME**-complete
Shy	**PTIME**-complete	**PTIME**-complete
Ward	**PTIME**-complete	**PTIME**-complete
Protected	**PTIME**-complete	**PTIME**-complete
Sticky-(Join)	$\mathbf{AC}_0$	**PTIME**-complete
Linear	$\mathbf{AC}_0$	**PTIME**-complete
Joinless	$\mathbf{AC}_0$	**PTIME**-complete
Inclusion-Dependencies	$\mathbf{AC}_0$	**PTIME**-complete

whereas, if $\mathbb{C} \supseteq \mathbf{PTIME}$, CERT-EVAL$[\textsf{Dyadic-}\mathcal{C}] \in \mathbf{PTIME}^{\mathbb{C}}$. To prove point 3 of the theorem, we observe that the membership follows from point 2 and from the fact that, for any deterministic class $\mathbb{C} \supseteq \mathbf{PTIME}$, it holds that $\mathbf{PTIME}^{\mathbb{C}} = \mathbb{C}$; whereas, the hardness derives from Proposition 5, since Dyadic-$\mathcal{C}$ includes the class $\mathcal{C}$, that is $\mathbb{C}$-hard by assumption. Finally, to prove point 4, we recall that by Theorem 6, Datalog $\subseteq$ Dyadic-$\mathcal{C}$; hence, since CERT-EVAL$[\textsf{Datalog}]$ is $\mathbf{PTIME}$-hard, the thesis follows.   □

Accordingly to the above theorem, immediately we get the following result.

Corollary 1
Complexity results in Table 2 do hold.

For studying the combined complexity, we need to take into account the fact that the database returned by Algorithm 2 (namely, D^+) is exponential with respect to the input one (namely, D). Indeed, the check $\mathbf{c} \in cert(q, D^+, \Sigma_\mathcal{C})$ performed by Algorithm 1 is done on an exponentially bigger database. Thus, in case CERT-EVAL$[\mathcal{C}]$ would have the same data complexity and combined complexity, it might happen that the combined complexity of CERT-EVAL$[\textsf{Dyadic-}\mathcal{C}]$ could be exponentially higher that the one of CERT-EVAL$[\mathcal{C}]$. Although all the considered classes in $\mathbb{E}^+_{syn}$ do not suffer from this shortcoming, before stating our general result, we need to focus on "well-behaved" classes of TGDs. A class $\mathcal{C}$ of TGDs enjoys the *dropping data-complexity* property if there is an exponential jump from the combined complexity of CERT-EVAL$[\mathcal{C}]$ to the data complexity of CERT-EVAL$[\mathcal{C}]$.

Proposition 6
Each class in $\mathbb{E}_{syn}$ enjoys the dropping data-complexity property.

We can now state the last result of the section, providing the combined complexity of problem CERT-EVAL over Dyadic-$\mathcal{C}$ sets of TGDs.

Theorem 8
Consider a class $\mathcal{C}$ of TGDs. In combined complexity, if CERT-EVAL$[\mathcal{C}]$ belongs to some decidable complexity class $\mathbb{C}$ and $\mathcal{C}$ enjoys the dropping data-complexity property, then the following hold:

1. If $\mathbb{C} \subseteq \textbf{EXPTIME}$, then CERT-EVAL[Dyadic-$\mathcal{C}$] is in $\textbf{EXPTIME}$;
2. If $\mathbb{C} \supseteq \textbf{EXPTIME}$, then CERT-EVAL[Dyadic-$\mathcal{C}$] is in $\textbf{EXPTIME}^{\mathbb{C}}$;
3. If $\mathbb{C} \supseteq \textbf{EXPTIME}$ is deterministic and CERT-EVAL[$\mathcal{C}$] is $\mathbb{C}$-complete, then it holds that CERT-EVAL[Dyadic-$\mathcal{C}$] is $\mathbb{C}$-complete too;
4. If $\mathcal{C} \supseteq$ Af-Inds, then CERT-EVAL[Dyadic-$\mathcal{C}$] is $\textbf{EXPTIME}$-hard.

Proof

The argument proceeds similarly to proof of Theorem 7 by arguing on Algorithm 3 to determine the complexity of CERT-EVAL[Dyadic-$\mathcal{C}$]. Let D be a database, $\Sigma \in$ Dyadic-$\mathcal{C}$ an ontology, $q(\mathbf{x})$ a CQ, and $\mathbf{c} \in \mathsf{C}^{|\mathbf{x}|}$ a tuple. Moreover, let $d = |consts(D)|$ and $\mu = \max_{P \in h\text{-}preds(\Sigma_{\text{HG}})} arity(P)$. As previously shown, Algorithm 3 invokes Algorithm 1, which in turn invokes Algorithm 2. Concerning the latter, by ignoring the computational costs of the oracle, it overall performs a number of step that is linear in $|\Sigma_{\text{HG}}| \cdot |h\text{-}preds(\Sigma_{\text{HG}})| \cdot d^{2\mu}$. Indeed, also in this case, this value is an upper bound for the number of calls to the oracle. This is enough to show point 2.

Concerning the memberships of point 1 and point 3, differently from the proof of Theorem 7, in combined complexity the maximum arity μ, the size of the sets Σ_{HG} and $\Sigma_{\mathcal{C}}$, as well as the size and the number of each query constructed at instruction 4 of Algorithm 2 are not bounded. Accordingly, also the size of the completed database returned by Algorithm 2 (namely D^{+}) may become exponential with respect to the input. More precisely, $|consts(D^{+})| = d$ and $|D^{+}| \leq |h\text{-}preds(\Sigma_{\text{HG}})| \cdot d^{\mu} + |D|$. Let n generically denote the size $||seq||$ of any sequence seq of objects given in input to CERT-EVAL[$\mathcal{C}$]. We can now consider the cost function $g(n)$ (resp., $f(n)$) of some algorithm/oracle that decides CERT-EVAL[$\mathcal{C}$] and shows that it belongs to $\mathbb{C}$ (resp., $\mathbb{C}_d$) in combined (resp., data) complexity. According to the dropping data-complexity property, we know that $g(n)$ grows at least exponentially faster than $f(n)$. Essentially, there is an exponential jump from $\mathbb{C}_d$ to $\mathbb{C}$ that does not depend on the size of the input database but only on the size of other parameters, namely the ontology, the query, and the tuple of constants. Consider now the query $q' := \langle \mathbf{x} \rangle \leftarrow \Phi(\mathbf{x}, \mathbf{y})$ constructed at instruction 4 of Algorithm 2 (we call it q' to avoid confusion with $q(\mathbf{x})$ mentioned at the beginning of this proof). At instruction 5 of the same algorithm, the oracle for CERT-EVAL[$\mathcal{C}$] checks whether $\mathbf{t} \in cert(q', D^{+}, \Sigma_{\mathcal{C}})$ holds. Since $g(n)$ grows at least exponentially faster than $f(n)$, we get that $g(||\mathbf{t}, q', D^{+}, \Sigma_{\mathcal{C}}||)$ remains of the same exponential order of $g(||\mathbf{t}, q', D, \Sigma_{\mathcal{C}}||)$, although $||D^{+}||$ may be exponentially larger than $||D||$.

Regarding point 1, if $\mathbb{C} \subseteq \textbf{EXPTIME}$, then we know that a $\mathbb{C}$-oracle for CERT-EVAL[$\mathcal{C}$] works at most exponentially in $||\mathbf{t}, q', \Sigma_{\mathcal{C}}||$ and at most polynomially (resp., exponentially) in $||D^{+}||$ (resp., $||D||$); thus, in this case, the $\mathbb{C}$-oracle cannot reach double-exponential time but it remains exponential. Therefore, CERT-EVAL[Dyadic-$\mathcal{C}$] $\in \textbf{EXPTIME}$.

For the memberships of point 3, we already know that CERT-EVAL[Dyadic-$\mathcal{C}$] is in $\textbf{EXPTIME}^{\mathbb{C}}$. Consider now a $\mathbb{C}$-oracle O for CERT-EVAL[$\mathcal{C}$] characterized by the cost function $g(n)$. If $\mathbb{C} \supseteq \textbf{EXPTIME}$ is deterministic, then O works with respect to $||\mathbf{t}, q', \Sigma_{\mathcal{C}}||$ in an exponentially faster way than with respect to $||D^{+}||$; thus, also in this case, O cannot exceed the power of $\mathbb{C}$. Therefore, $\textbf{EXPTIME}^{\mathbb{C}}$, in a sense, collapses to $\mathbb{C}$.

Finally, we conclude the proof by considering the hardness of points 3 and 4. In the first case, we observe that it derives from Proposition 5, since Dyadic-$\mathcal{C}$ includes the

Table 3. *Combined complexity comparison of* CERT-EVAL[$\mathcal{C}$] *with* CERT-EVAL[*Dyadic*-$\mathcal{C}$]

Class $\mathcal{C}$	$\mathcal{C}$	Dyadic-$\mathcal{C}$
Weakly-(Fr)-Guarded	**2EXPTIME**-complete	**2EXPTIME**-complete
(Fr)-Guarded	**2EXPTIME**-complete	**2EXPTIME**-complete
Weakly-Acyclic	**2EXPTIME**-complete	**2EXPTIME**-complete
Jointly-Acyclic	**2EXPTIME**-complete	**2EXPTIME**-complete
Datalog	**EXPTIME**-complete	**EXPTIME**-complete
Shy	**EXPTIME**-complete	**EXPTIME**-complete
Ward	**EXPTIME**-complete	**EXPTIME**-complete
Protected	**EXPTIME**-complete	**EXPTIME**-complete
Sticky-(Join)	**EXPTIME**-complete	**EXPTIME**-complete
Linear	**PSPACE**-complete	**EXPTIME**-complete
Joinless	**PSPACE**-complete	**EXPTIME**-complete
Inclusion-Dependencies	**PSPACE**-complete	**EXPTIME**-complete

class $\mathcal{C}$, that is $\mathbb{C}$-hard by assumption. For point 4, we recall that by Theorem 6, Datalog $\subseteq$ Dyadic-$\mathcal{C}$; hence, since CERT-EVAL[Datalog] is **EXPTIME**-hard, it follows the thesis. □

The following immediately derives from above theorem.

Corollary 2
Complexity results in Table 3 do hold.

7 Conclusion

Dyadic decomposable sets form a novel decidable class of TGDs that encompasses and generalizes all the existing (syntactic and semantic) decidable classes of TGDs. In the near feature, it would be interesting to implement a prototype for dyadic existential rules by exploiting different kinds of existing reasoners.

Acknowledgments

References

ABITEBOUL, S., HULL, R. AND VIANU, V. 1995. *Foundations of Databases*, vol. 8. Addison-Wesley Reading.

BAADER, F., CALVANESE, D., McGUINNESS, D. L., NARDI, D. AND PATEL-SCHNEIDER, P. F. Eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

BAGET, J., LECLÈRE, M., MUGNIER, M. AND SALVAT, E. 2009. Extending decidable cases for rules with existential variables. In *IJCAI*, 677–682.

BAGET, J., LECLÈRE, M., MUGNIER, M. AND SALVAT, E. 2011. On rules with existential variables: Walking the decidability line. *Artificial Intelligence 175*, 9-10, 1620–1654.

BALDAZZI, T., BELLOMARINI, L., FAVORITO, M. AND SALLINGER, E. 2022. On the relationship between shy and warded datalog+/-. CoRR, abs/2202.06285.

BERGER, G., GOTTLOB, G., PIERIS, A. AND SALLINGER, E. 2022. *The Space-Efficient Core of Vadalog 2022*, vol. 47, 1:1–1:46.

CALÌ, A., GOTTLOB, G. AND KIFER, M. 2008. Taming the infinite chase: Query answering under expressive relational constraints. In *KR*. AAAI Press, 70–80.

CALÌ, A., GOTTLOB, G. AND KIFER, M. 2013. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research 48*, 115–174.

CALÌ, A., GOTTLOB, G. AND LUKASIEWICZ, T. 2012a. A general datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics 14*a, 57–83.

CALÌ, A., GOTTLOB, G. AND PIERIS, A. 2010. Advanced processing for ontological queries. *Proceedings of the VLDB Endowment 3*, 1-2, 554–565.

CALÌ, A., GOTTLOB, G. AND PIERIS, A. 2012b. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence 193*b, 87–128.

CERI, S., GOTTLOB, G. AND TANCA, L. 1989. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering 1*, 1, 146–166.

DEUTSCH, A., NASH, A. AND REMMEL, J. B. The chase revisited. In *PODS* 2008. ACM, 149–158.

FAGIN, R., KOLAITIS, P. G., MILLER, R. J. AND POPA, L. 2005. Data exchange: Semantics and query answering. *Theoretical Computer Science 336*, 1, 89–124.

GOGACZ, T. AND MARCINKOWSKI, J. *Converging to the Chase - A Tool for Finite Controllability*, vol. 83, 180–206.

GOTTLOB, G., MANNA, M. AND MARTE, C. Dyadic existential rules. In *Datalog*, CEUR Workshop Proceedings, vol. 3203. CEUR-WS.org, 83–96.

GOTTLOB, G. AND PIERIS, A. Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue. In *IJCAI*. AAAI Press, 2999–3007.

JOHNSON, D. S. AND KLUG, A. C. 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences 28*, 1, 167–189.

KRÖTZSCH, M. AND RUDOLPH, S. Extending decidable existential rules by joining acyclicity and guardedness. In *IJCAI*. IJCAI/AAAI, 963–968.

LEONE, N., MANNA, M., TERRACINA, G. AND VELTRI, P. 2019. Fast query answering over existential rules. *ACM Transactions on Computational Logic 20*, 2, 12:1–12:48.