Journal of Artificial Intelligence Research 48 (2013) 115–174 Submitted 11/12; published 10/13

Taming the Infinite Chase: Query Answering under Expressive Relational Constraints

Andrea Calì ANDREA@DCS.BBK.AC.UK
Department of Computer Science and Information Systems
University of London, Birkbeck College, UK

Georg Gottlob GEORG.GOTTLOB@CS.OX.AC.UK
Department of Computer Science
University of Oxford, UK

Michael Kifer KIFER@CS.STONYBROOK.EDU
Department of Computer Science
Stony Brook University, USA

Abstract

The *chase* algorithm is a fundamental tool for query evaluation and for testing query containment under *tuple-generating dependencies (TGDs)* and *equality-generating dependencies (EGDs)*. So far, most of the research on this topic has focused on cases where the chase procedure terminates. This paper introduces expressive classes of TGDs defined via syntactic restrictions: *guarded TGDs (GTGDs)* and *weakly guarded sets of TGDs (WGTGDs)*. For these classes, the chase procedure is not guaranteed to terminate and thus may have an infinite outcome. Nevertheless, we prove that the problems of conjunctive-query answering and query containment under such TGDs are decidable. We provide decision procedures and tight complexity bounds for these problems. Then we show how EGDs can be incorporated into our results by providing conditions under which EGDs do not harmfully interact with TGDs and do not affect the decidability and complexity of query answering. We show applications of the aforesaid classes of constraints to the problem of answering conjunctive queries in *F-Logic Lite*, an object-oriented ontology language, and in some tractable Description Logics.

1. Introduction

This paper studies a simple yet fundamental rule-based language for ontological reasoning and query answering: the language of *tuple-generating dependencies (TGDs)*. This formalism captures a wide variety of logics that so far were considered unrelated to each other: the OWL-based languages $\mathcal{EL}$ (Baader, Brandt, & Lutz, 2005) and DL-Lite (Calvanese, De Giacomo, Lembo, Lenzerini, & Rosati, 2007; Artale, Calvanese, Kontchakov, & Zakharyaschev, 2009) on the one hand and object-based languages like F-Logic Lite (Calì & Kifer, 2006) on the other. The present paper is a significant extension of our earlier work (Calì, Gottlob, & Kifer, 2008), which has since been applied in other contexts and gave rise to the Datalog$^{\pm}$ family (Calì, Gottlob, & Pieris, 2011) of ontology languages. The present paper focuses on the fundamental complexity results underlying one of the key fragments of this family. Subsequent work has focused on the study of various special cases of this formalism (Calì, Gottlob, & Lukasiewicz, 2012a), their complexity, and extensions based on other paradigms (Calì, Gottlob, & Pieris, 2012b).

Our work is also closely related to the work on query answering and query containment (Chandra & Merlin, 1977), which are central problems in database theory and knowledge representation and, in most cases, are reducible to each other. They are especially interesting in the presence of integrity constraints—or *dependencies*, in database parlance. In databases, query containment has been used for query optimization and schema integration (Aho, Sagiv, & Ullman, 1979; Johnson & Klug, 1984; Millstein, Levy, & Friedman, 2000), while in knowledge representation it is often used for object classification, schema integration, service discovery, and more (Calvanese, De Giacomo, & Lenzerini, 2002; Li & Horrocks, 2003).

A practically relevant instance of the containment problem was first studied by Johnson and Klug (1984) for functional and inclusion dependencies and later by Calvanese, De Giacomo, and Lenzerini (1998). Several additional decidability results were obtained by focusing on concrete applications. For instance, the work by Calì and Martinenghi (2010) considers constraints arising from Entity-Relationship diagrams, while that by Calì and Kifer (2006) considers constraints derived from a relevant subset of F-logic (Kifer, Lausen, & Wu, 1995), called F-Logic Lite.

Some literature studies variants or subclasses of *tuple-generating dependencies (TGDs)* for the purpose of reasoning and query answering. A TGD is a Horn-like rule with existentially-quantified variables in the head. Some early works on this subject dubbed the resulting language *Datalog with value invention* (Mailharrow, 1998; Cabibbo, 1998). More formally, a TGD $\forall \mathbf{X} \forall \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$ is a first-order formula, where $\Phi(\mathbf{X}, \mathbf{Y})$ and $\Psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms, called *body* and *head* of the TGD, respectively. A TGD is satisfied by a relational instance B if whenever the body of the TGD is satisfied by B then B also satisfies the head of the TGD. It is possible to enforce a TGD that is *not* satisfied by adding new facts to B so that the head, and thus the TGD itself, will become satisfied. These new facts will contain *labeled null values* (short: *nulls*) in the positions corresponding to variables $\mathbf{Z}$. Such nulls are similar to Skolem constants. The *chase* of a database D in the presence of a set Σ of TGDs is the process of iterative enforcement of all dependencies in Σ, until a fixpoint is reached. The result of such a process, which we also call *chase*, can be infinite and, in this case, this procedure cannot be used without modifications in decision algorithms. Nevertheless, the result of a chase serves as a fundamental theoretical tool for answering queries in the presence of TGDs (Calì, Lembo, & Rosati, 2003a; Fagin, Kolaitis, Miller, & Popa, 2005) because it is representative of all models of $D \cup \Sigma$.

In the present paper, we do not focus on a specific logical theory. Instead, we tackle the common issue of the possibly non-terminating chase underlying several of the earlier studies, including the works by Johnson and Klug (1984), by Calì and Martinenghi (2010), and by Calì and Kifer (2006). All these works study constraints in the language of TGDs and *equality-generating dependencies (EGDs)* using the chase technique, and all face the problem that the chase procedure might generate an infinite result. We deal with this problem in a much more general way by carving out a very large class of constraints for which the infinite chase can be "tamed", i.e., modified so that it would become a decision procedure for query answering.

In Section 3, we define the notions of sets of *guarded TGDs (GTGDs)* and of *weakly guarded sets of TGDs (WGTGDs)*. A TGD is guarded if its body contains an atom called *guard* that covers all variables occurring in the body. WGTGDs generalize guarded TGDs by requiring guards to cover only the variables occurring at so-called *affected* positions (predicate positions that may contain some labeled nulls generated during the chase). Note that *inclusion dependencies* (or IDs) can be viewed as trivially guarded TGDs. The importance of guards lies in Theorem 3.5, which shows that there is a fixed set Σ_u of GTGDs plus a single non-guarded TGD, such that query evaluation under Σ_u is undecidable. However, we show that for WGTGDs the (possibly infinite) result of the chase has finite treewidth (Theorem 3.14). We then use this result together with well-known results about the generalized tree-model property (Goncalves & Grädel, 2000; Grädel, 1999) to show that evaluating Boolean conjunctive queries is decidable for WGTGDs (and thus also for GTGDs). Unfortunately, this result does not directly provide useful complexity bounds.

In Section 4, we show lower complexity bounds for conjunctive query answering under weakly guarded sets of TGDs. We prove, by Turing machine simulations, that query evaluation under weakly guarded sets of TGDs is EXPTIME-hard in case of a fixed set of TGDs, and 2EXPTIME-hard in case the TGDs are part of the input.

In Section 5, we address upper complexity bounds for query answering under weakly guarded sets of TGDs. Let us first remark that showing $D \cup \Sigma \models Q$ is equivalent to showing that the theory $\mathcal{T} = D \cup \Sigma \cup \{\neg Q\}$ is unsatisfiable. Unfortunately, $\mathcal{T}$ is in general not guarded because Q is not and because WGTGDs are generally non-guarded first-order sentences (while GTGDs are). Therefore, we cannot (as one might think at first glance) directly use known results on guarded logics (Goncalves & Grädel, 2000; Grädel, 1999) to derive complexity results for query evaluation. We thus develop completely new algorithms by which we prove that the problem in question is EXPTIME-complete in case of bounded predicate arities and, even in case the TGDs are fixed, 2EXPTIME-complete in general.

In Section 6, we derive complexity results for reasoning with GTGDs. In the general case, the complexity is as for WGTGDs but, interestingly, when reasoning with a *fixed* set of dependencies (which is the usual setting in data exchange and in description logics), we get much better results: evaluating Boolean queries is NP-complete and is in PTIME in case the query is atomic. Recall that Boolean query evaluation is NP-hard even in case of a simple database without integrity constraints (Chandra & Merlin, 1977). Therefore, the above NP upper bound for general Boolean queries is optimal, i.e., there is no class of TGDs for which query evaluation (or query containment) is more efficient.

In Section 7, we describe a semantic condition on weakly guarded sets of TGDs. We prove that whenever a set of WGTGDs fulfills this condition, answering Boolean queries is in NP, and answering atomic queries, as well as queries of bounded treewidth, is in PTIME.

Section 8 extends our results to the case of TGDs with multiple-atom heads. The extension is trivial for all cases except for the case of bounded predicate arity.

Section 9 deals with *equality generating dependencies (EGDs)*, a generalization of functional dependencies. Unfortunately, as shown in works by Chandra and Vardi (1985), Mitchell (1983), Johnson and Klug (1984), Koch (2002), and Calì et al. (2003a), query answering and many other problems become undecidable in case we admit both TGDs and EGDs. It remains undecidable even if we mix the simplest class of guarded TGDs, namely, inclusion dependencies, with the simplest type of EGDs, namely functional dependencies, and even key dependencies (Chandra & Vardi, 1985; Mitchell, 1983; Johnson & Klug, 1984; Calì et al., 2003a). In Section 9, we present a sufficient semantic condition for decidabil-

BCQ type	GTGDs	WGTGDs
general	2EXPTIME	2EXPTIME
atomic or fixed	2EXPTIME	2EXPTIME

Query answering for variable TGDs.

BCQ type	GTGDs	WGTGDs
general	NP	EXPTIME
atomic or fixed	PTIME	EXPTIME

Query answering for fixed TGDs.

BCQ type	GTGDs	WGTGDs
general	EXPTIME	EXPTIME
atomic or fixed	EXPTIME	EXPTIME

Query answering for fixed predicate arity.

Figure 1: Summary of results. All complexity bounds are tight.

ity of query-answering under sets of TGDs and general EGDs. We call EGDs *innocuous* when, roughly speaking, their application (i.e., enforcement) does not introduce new atoms, but only eliminates atoms. We show that innocuous EGDs can be essentially *ignored* for conjunctive query evaluation and query containment testing.

The TGD-based ontology languages in this paper are part of the larger family of ontology languages called Datalog$^\pm$ (Calì et al., 2011). Our results subsume the main decidability and NP-complexity result by Johnson and Klug (1984), the decidability and complexity results on F-Logic Lite by Calì and Kifer (2006), and those on DL-Lite as special cases. In fact, Section 10 shows that our results are even more general than that.

The complexity results of this paper, together with some of their immediate consequences, are summarized in Figure 1, where all complexity bounds are tight. Notice that the complexity in the case of fixed queries *and* fixed TGDs is the so-called *data complexity*, i.e., the complexity with respect to the data only, which is of particular interest in database applications. The complexity for variable Boolean conjunctive queries (BCQs) and variable TGDs is called *combined complexity*. It is easy to see (but we will not prove it formally for all classes) that all complexity results for atomic or fixed queries extend to queries of bounded width, where by *width* we mean treewidth or even hypertree width (Gottlob, Leone, & Scarcello, 2002)—see also the works by Adler, Gottlob, and Grohe (2007), and by Gottlob, Leone, and Scarcello (2001).

2. Preliminaries

In this section we define the basic notions that we use throughout the paper.

2.1 Relations, Instances and Queries

A *relational schema* $\mathcal{R}$ is a set of relational predicates, each having an *arity*—a non-negative integer that represents the number of arguments the predicate takes. We write r/n to say that a relational predicate r has arity n. Given an n-ary predicate $r \in \mathcal{R}$, a *position* $r[k]$, where $1 \leqslant k \leqslant n$, refers to the k-th argument of r. We will assume an underlying relational schema $\mathcal{R}$ and postulate that all queries and constraints use only the predicates in $\mathcal{R}$. The schema $\mathcal{R}$ will sometimes be omitted when it is clear from the context or is immaterial.

We introduce the following pairwise disjoint sets of symbols: *(i)* A (possibly infinite) set Δ of data *constants*, which constitute the "normal" domain of the databases over the schema $\mathcal{R}$; *(ii)* a set Δ_N of *labeled nulls*, i.e., "fresh" Skolem constants; and *(iii)* an infinite set Δ_V of variables, which are used in queries and constraints. Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. We also assume a lexicographic order on $\Delta \cup \Delta_N$, with every labeled null in Δ_N following all constant symbols in Δ. Sets of variables (or sequences, when the order is relevant) will be denoted by $\mathbf{X}$, i.e., $\mathbf{X} = X_1, \ldots, X_k$, for some k. The notation $\exists\mathbf{X}$ is a shorthand for $\exists X_1 \ldots \exists X_k$, and similarly for $\forall\mathbf{X}$.

An *instance* of a relational predicate r/n is a (possibly infinite) set of atomic formulas (atoms) of the form $r(c_1, \ldots, c_n)$, where $\{c_1, \ldots, c_n\} \subseteq \Delta \cup \Delta_N$. Such atoms are also called *facts*. When the fact $r(c_1, \ldots, c_n)$ is true, we say that the *tuple* $\langle c_1, \ldots, c_n \rangle$ belongs to the instance of r (or just that it is in r, if confusion does not arise). An instance of the relational schema $\mathcal{R} = \{r_1, \ldots, r_m\}$ is the set comprised of the instances of $r_1, \ldots, r_m$. When instances are treated as first-order formulas, each labeled null is viewed as an existential variable with the same name, and relational instances with nulls correspond to a conjunction of atoms preceded by the existential quantification of all the nulls. For instance, $\{r(a, z_1, z_2, z_1), s(b, z_2, z_3)\}$, where $\{z_1, z_2, z_3\} \subseteq \Delta_N$ and $\{a, b\} \subseteq \Delta$, is expressed as $\exists z_1 \exists z_2 \exists z_3\, r(a, z_1, z_2, z_1) \wedge s(b, z_2, z_3)$. In the following, we will omit these quantifiers.

A fact $r(c_1, \ldots, c_n)$ is said to be *ground* if $c_i \in \Delta$ for all $i \in \{1, \ldots, n\}$. In such a case, also the tuple $\langle c_1, \ldots, c_n \rangle$ is said to be ground. A relation or schema instance all of whose facts are ground is said to be ground, and a ground instance of $\mathcal{R}$ is also called a *database*.

If A is a *sequence* of atoms $\langle \underline{a}_1, \ldots, \underline{a}_k \rangle$ or a *conjunction* of atoms $\underline{a}_1 \wedge \ldots \wedge \underline{a}_k$, we use *atoms*$(A)$ to denote the *set* of the atoms in A: *atoms*$(A) = \{\underline{a}_1, \ldots, \underline{a}_k\}$. Given a (ground or non-ground) atom $\underline{a}$, the *domain* of $\underline{a}$, denoted by $dom(\underline{a})$, is the set of all values (variables, constants or labeled nulls) that appear as arguments in $\underline{a}$. If A is a set of atoms, we define $dom(A) = \bigcup_{\underline{a} \in A} dom(\underline{a})$. If A is a sequence or a conjunction of atoms then we define $dom(A) = dom(atoms(A))$. If A is an atom, a set, a sequence, or a conjunction of atoms, we write $vars(A)$ to denote the set of variables in A.

Given an instance B of a relational schema $\mathcal{R}$, the *Herbrand Base* of B, denoted $HB(B)$, is the set of all atoms that can be formed using the predicate symbols of $\mathcal{R}$ and arguments in $dom(B)$. Notice that this is an extension of the classical notion of Herbrand Base, which includes ground atoms only.

An n-ary *conjunctive query (CQ)* over $\mathcal{R}$ is a formula of the form $q(X_1, \ldots, X_n) \leftarrow \Phi(\mathbf{X})$, where q is a predicate not appearing in $\mathcal{R}$, all the variables $X_1, \ldots, X_n$ appear in $\mathbf{X}$, and $\Phi(\mathbf{X})$, called the *body* of the query, is a conjunction of atoms constructed with predicates from $\mathcal{R}$. The arity of a query is the arity of its head predicate q. If q has arity 0, then the conjunctive query is called *Boolean* (BCQ). For BCQs, it is convenient to drop the head predicate and simply view the query as the *set* of atoms in $\Phi(\mathbf{X})$. If not stated otherwise, we assume that queries contain no constants, since constants can be eliminated from queries by a simple polynomial time transformation. We will also sometimes refer to conjunctive queries by just "queries". The *size* of a conjunctive query Q is denoted by $|Q|$; it represents the number of atoms in Q.

2.2 Homomorphisms

A *mapping* from a set of symbols S_1 to another set of symbols S_2 can be seen as a function $\mu : S_1 \to S_2$ defined as follows: *(i)* $\varnothing$ (the empty mapping) is a mapping; *(ii)* if μ is a mapping, then $\mu \cup \{X \to Y\}$, where $X \in S_1$ and $Y \in S_2$ is a mapping if μ does not already contain some $X \to Y'$ with $Y \neq Y'$. If $X \to Y$ is in a mapping μ, we write $\mu(X) = Y$. The notion of a mapping is naturally extended to atoms as follows. If $\underline{a} = r(c_1, \ldots, c_n)$ is an atom and μ a mapping, we define $\mu(\underline{a}) = r(\mu(c_1), \ldots, \mu(c_n))$. For a *set* of atoms, $A = \{\underline{a}_1, \ldots, \underline{a}_m\}$, $\mu(A) = \{\mu(\underline{a}_1), \ldots, \mu(\underline{a}_m)\}$. The set of atoms $\mu(A)$ is also called *image* of A with respect to μ. For a *conjunction* of atoms $C = \underline{a}_1 \wedge \ldots \wedge \underline{a}_m$, $\mu(C)$ is a shorthand for $\mu(\mathit{atoms}(C))$, that is, $\mu(C) = \{\mu(\underline{a}_1), \ldots, \mu(\underline{a}_m)\}$.

A *homomorphism* from a set of atoms A_1 to another set of atoms A_2, with $\mathit{dom}(A_1 \cup A_2) \subseteq \Delta \cup \Delta_N \cup \Delta_V$ is a mapping μ from $\mathit{dom}(A_1)$ to $\mathit{dom}(A_2)$ such that the following conditions hold: *(1)* if $c \in \Delta$ then $\mu(c) = c$; *(2)* $\mu(A_1) \subseteq A_2$, i.e., if an atom, $\underline{a}$, is in A_1, then the atom $\mu(\underline{a})$ is in A_2. In this case, we will say that A_1 *maps* to A_2 via μ.

The answer to a conjunctive query Q of the form $q(X_1, \ldots, X_n) \leftarrow \Phi(\mathbf{X})$ over an instance B of $\mathcal{R}$, denoted by $Q(B)$, is defined as follows: a tuple $\mathbf{t} \in (\Delta \cup \Delta_N)^n$, is in $Q(B)$ iff there is a homomorphism μ that maps $\Phi(\mathbf{X})$ to atoms of B, and $\langle X_1, \ldots, X_n \rangle$ to $\mathbf{t}$. In this case, by abuse of notation, we also write $q(\mathbf{t}) \in Q(B)$. A Boolean conjunctive query Q has a *positive* answer on B iff $\langle\rangle$ (the tuple with no elements) is in $Q(B)$; otherwise, it is said to have a *negative* answer.

2.3 Relational Dependencies

We now define the main type of dependencies used in this paper, the *tuple-generating dependencies*, or *TGDs*.

Definition 2.1. Given a relational schema $\mathcal{R}$, a TGD σ over $\mathcal{R}$ is a first-order formula of the form $\forall \mathbf{X} \forall \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ and $\Psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called *body* and *head* of the TGD, respectively; they are denoted by $\mathit{body}(\sigma)$ and $\mathit{head}(\sigma)$. Such a dependency is satisfied in an instance B of $\mathcal{R}$ if, whenever there is a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of B, there exists an *extension* h_2 of h (i.e., $h_2 \supseteq h$) that maps the atoms of $\Psi(\mathbf{X}, \mathbf{Z})$ to atoms of B.

To simplify the notation, we will usually omit the universal quantifiers in TGDs. We will also sometimes call TGDs *rules* because of the implication symbol in them. Notice that, in general, constants of Δ can appear not only in the body, but also in the heads of TGDs. For simplicity and without loss of generality, we assume that all constants that appear in the the head of a TGDs also appear in the body of the same TGD.

The symbol $\models$ will be used henceforth for the usual logical entailment, where sets of atoms and TGDs are viewed as first-order theories. For such theories, we do not restrict ourselves to finite models: we consider arbitrary models that could be finite or infinite. This aspect is further discussed in Section 11.

2.4 Query Answering and Containment under TGDs

We now define the notion of *query answering* under TGDs. A similar notion is used in data exchange (Fagin et al., 2005; Gottlob & Nash, 2006) and in query answering over incomplete data (Calì et al., 2003a). Given a database that does not satisfy all the constraints in Σ, we first define the set of completions (or *repairs*—see Arenas, Bertossi, & Chomicki, 1999) of that database, which we call *solutions*.

Definition 2.2. Consider a relational schema $\mathcal{R}$, a set of TGDs Σ, and a database D for $\mathcal{R}$. The set of instances $\{B \mid B \models D \cup \Sigma\}$ is called the set of *solutions* of D given Σ, and is denoted by $\mathrm{sol}(D, \Sigma)$.

The following is the definition of the problem, which we denote by CQAns, of answering conjunctive queries under TGDs. The answers defined here are also referred to as *certain answers* (see Fagin et al., 2005).

Definition 2.3. Consider a relational schema $\mathcal{R}$, a set of TGDs Σ, a database D for $\mathcal{R}$, and a conjunctive query Q on $\mathcal{R}$. The *answer* to a conjunctive query Q on D given Σ, denoted by $ans(Q, D, \Sigma)$, is the set of tuples $\mathbf{t}$ such that for every $B \in \mathrm{sol}(D, \Sigma)$, $\mathbf{t} \in Q(B)$ holds.

Notice that the components of $\mathbf{t}$ in the above definition are necessarily constants from Δ. When $\mathbf{t} \in ans(Q, D, \Sigma)$, we also write $D \cup \Sigma \cup \{Q\} \models q(\mathbf{t})$, where Q is represented as a rule $body(Q) \rightarrow q(\mathbf{X})$.

Containment of queries over relational databases has long been considered a fundamental problem in query optimization, especially query containment under constraints such as TGDs. Below we formally define this problem, which we call CQCont.

Definition 2.4. Consider a relational schema $\mathcal{R}$, a set Σ of TGDs on $\mathcal{R}$, and two conjunctive queries Q_1, Q_2 expressed over $\mathcal{R}$. We say that Q_1 *is contained in* Q_2 *under* Σ, denoted by $Q_1 \subseteq_\Sigma Q_2$, if for every instance B for $\mathcal{R}$ such that $B \models \Sigma$ we have $Q_1(B)$ is a subset of $Q_2(B)$.

2.5 The Chase

The *chase* was introduced as a procedure for testing implication of dependencies (Maier, Mendelzon, & Sagiv, 1979), but later also employed for checking query containment (Johnson & Klug, 1984) and query answering on incomplete data under relational dependencies (Calì et al., 2003a). Informally, the chase procedure is a process of repairing a database with respect to a set of dependencies, so that the result of the chase satisfies the dependencies. By "chase" we may refer either to the chase procedure or to its output. The chase works on a database through the so-called TGD *chase rule*, which defines the result of the applications of a TGD and comes in two flavors: *oblivious* and *restricted*.

Definition 2.5. [Oblivious Applicability] Consider an instance B of a schema $\mathcal{R}$, and a TGD $\sigma = \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$. We say that σ is *obliviously applicable* to B if there exists a homomorphism h such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq B$.

Definition 2.6. [Restricted Applicability] Consider an instance B of a schema $\mathcal{R}$, and a TGD $\sigma = \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$. We say that σ is *restrictively applicable* to B if there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq B$, but there is *no* extension h' of $h|_{\mathbf{X}}$ such that $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq B$.[1]

The oblivious form of applicability is called this way because it "forgets" to check whether the TGD is already satisfied. In contrast, a TGD is restrictively applicable only if it is not already satisfied.

Definition 2.7. [TGD Chase Rule] Let σ be a TGD of the form $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$ and suppose that it is obliviously (resp., restrictively) applicable to an instance B via a homomorphism h. Let h' be an extension of $h|_{\mathbf{X}}$ such that, for each $Z \in \mathbf{Z}$, $h'(Z)$ is a "fresh" labeled null of Δ_N not occurring in B, and following lexicographically all those in B. The result of the *oblivious* (resp., *restricted*) *application* of σ on B with h is $B' = B \cup h'(\Psi(\mathbf{X}, \mathbf{Z}))$. We write $B \xrightarrow{\sigma,h}_O B'$ (resp., $B \xrightarrow{\sigma,h}_R B'$) to denote that B' is obtained from B through a single oblivious (resp., restricted) chase step.

The TGD chase rule, defined above, is the basic building block to construct the chase of a database under a set of TGDs. Depending on the notion of applicability in use—oblivious or restricted—we get the oblivious or the restricted chase. The formal definition of the chase is given below.

Definition 2.8. [Oblivious and Restricted Chase] Let D be a database and Σ a set of TGDs. An *oblivious (resp., restricted) chase sequence* of D with respect to Σ is a sequence of instances $B_0, B_1, B_2, \ldots$ such that $B_0 = D$ and, for all $i \geqslant 0$, $B_i \xrightarrow{\sigma_i,h_i}_O B_{i+1}$ (resp., $B_i \xrightarrow{\sigma_i,h_i}_R B_{i+1}$) and $\sigma_i \in \Sigma$. We also assume that in any chase sequence the same pair $\langle \sigma_i, h_i \rangle$ is never applied more than once. The oblivious (resp., restricted) chase of D with respect to Σ, denoted $\mathit{Ochase}(D, \Sigma)$ (resp., $\mathit{Rchase}(D, \Sigma)$), is defined as follows:

- A *finite oblivious (resp., restricted) chase* of D with respect to Σ is a finite oblivious (resp., restricted) chase sequence $B_0, \ldots, B_m$ such that $B_i \xrightarrow{\sigma_i,h_i}_O B_{i+1}$ (resp., $B_i \xrightarrow{\sigma_i,h_i}_R B_{i+1}$) for all $0 \leqslant i < m$, and there is no $\sigma \in \Sigma$ such that its application yields an instance $B' \neq B_m$. We define $\mathit{Ochase}(D, \Sigma) = B_m$ (resp., $\mathit{Rchase}(D, \Sigma) = B_m$).
- An *infinite oblivious (resp., restricted) chase sequence* $B_0, B_1, \ldots$, where $B_i \xrightarrow{\sigma_i,h_i}_O B_{i+1}$ (resp., $B_i \xrightarrow{\sigma_i,h_i}_R B_{i+1}$) for all $i \geqslant 0$, is *fair* if whenever a TGD $\sigma = \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$ of Σ is obliviously (resp., restrictedly) applicable to B_i with homomorphism h, then there exists an extension h' of $h|_{\mathbf{X}}$ and $k > 0$ such that $h'(\mathit{head}(\sigma)) \subseteq B_k$. An *infinite oblivious chase* of D with respect to Σ is a fair infinite chase sequence $B_0, B_1, \ldots$ such that $B_i \xrightarrow{\sigma_i,h_i}_O B_{i+1}$ (resp., $B_i \xrightarrow{\sigma_i,h_i}_R B_{i+1}$) for all $i \geqslant 0$. In this case, we define $\mathit{Ochase}(D, \Sigma) = \lim_{i \to \infty} B_i$ (resp., $\mathit{Rchase}(D, \Sigma) = \lim_{i \to \infty} B_i$).

It is easy to see that the chase can be infinite, if the sequence of applications of the chase rule is infinite. We remark that the chase was defined for databases of ground tuples. However, the definition straightforwardly applies also to arbitrary instances, possibly containing

1. $h|_{\mathbf{X}}$ denotes the restriction of h to the set of variables of $\mathbf{X}$.

labeled nulls. We assume a fair deterministic strategy for constructing chase sequences. We use $Ochase^{[i]}(D,\Sigma)$ (resp., $Rchase^{[i]}(D,\Sigma)$) to denote the result of the i-th step of the oblivious (resp., restricted) chase of D with respect to Σ. Notice that $Ochase^{[i]}(D,\Sigma)$ (resp., $Rchase^{[i]}(D,\Sigma)$) is called the oblivious (resp., restricted) chase of D with respect to Σ *up to the derivation level* i, as in the work by Calì et al. (2012a).

Example 2.9. In this example, we show an oblivious chase procedure. Consider the following set $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ of TGDs.

$$\begin{array}{lrcl}
\sigma_1: & r_3(X,Y) & \to & r_2(X) \\
\sigma_2: & r_1(X,Y) & \to & \exists Z\, r_3(Y,Z) \\
\sigma_3: & r_1(X,Y), r_2(Y) & \to & \exists Z\, r_1(Y,Z) \\
\sigma_4: & r_1(X,Y) & \to & r_2(Y)
\end{array}$$

and let $D = \{r_1(a,b)\}$. The chase procedure adds to D the following sequence of atoms: $r_3(b,z_1)$ via σ_2, $r_2(b)$ via σ_4, $r_1(b,z_2)$ via σ_3, $r_3(z_2,z_3)$ via σ_2, $r_2(z_2)$ via σ_4, and so on. $\square$

2.6 Query Answering and the Chase

The problems of query containment and answering under TGDs are closely related to each other and to the notion of chase, as explained below.

Theorem 2.10 (see Nash, Deutsch, & Remmel, 2006)**.** *Consider a relational schema $\mathcal{R}$, a database D for $\mathcal{R}$, a set Σ of TGDs on $\mathcal{R}$, an n-ary conjunctive query Q with head-predicate q, and an n-ary ground tuple $\mathbf{t}$ (with values in Δ). Then $\mathbf{t} \in ans(Q,D,\Sigma)$ iff there exists a homomorphism h such that $h(body(Q)) \subseteq Rchase(D,\Sigma)$ and $h(head(Q)) = q(\mathbf{t})$.*

Notice that the fact that $h(body(Q)) \subseteq Rchase(D,\Sigma)$ and $h(head(Q)) = q(\mathbf{t})$ is equivalent to saying that $q(\mathbf{t}) \in Q(Rchase(D,\Sigma))$, or that $Rchase(D,\Sigma) \cup \{Q\} \models q(\mathbf{t})$. The result of Theorem 2.10 is important, and it holds because the (possibly infinite) restricted chase is a *universal solution* (Fagin et al., 2005), i.e., a representative of all instances in $\mathrm{sol}(D,\Sigma)$. More formally, a universal solution for D under Σ is a (possibly infinite) instance U such that, for every instance $B \in \mathrm{sol}(D,\Sigma)$, there exists a homomorphism that maps U to B. In the work by Nash et al. (2006) it is shown that the chase constructed with respect to TGDs is a universal solution.

A *freezing homomorphism* for a query is a homomorphism that maps every distinct variable in the query into a distinct labeled null in Δ_N. The following well known result is a slight extension of a result by Chandra and Merlin (1977).

Theorem 2.11. *Consider a relational schema $\mathcal{R}$, a set Σ of TGDs on $\mathcal{R}$, and two conjunctive queries Q_1, Q_2 on $\mathcal{R}$. Then $Q_1 \subseteq_\Sigma Q_2$ iff $\lambda(head(Q_1)) \in Q_2(Rchase(\lambda(body(Q_1)),\Sigma)$ for some freezing homomorphism λ for Q_1.*

From this and the results by Johnson and Klug (1984) and by Nash et al. (2006), we easily obtain the following result, which is considered folklore.

Corollary 2.12. *The problems* CQAns *and* CQCont *are mutually* LOGSPACE*-reducible.*

2.7 Oblivious vs. Restricted Chase

As observed by Johnson and Klug (1984) in the case of functional and inclusion dependencies, things are more complicated if the restricted chase is used instead of the oblivious one, since applicability of a TGD depends on the presence of other atoms previously added to the database by the chase. It is technically easier to use the oblivious chase and it can be used in lieu of the restricted chase because, as we shall prove now, a result similar to Theorem 2.10 holds for the oblivious chase, i.e., it is also universal. This result, to the best of our knowledge, has never been explicitly stated before. For the sake of completeness, we present a full proof here.

Theorem 2.13. *Consider a set Σ of TGDs on a relational schema $\mathcal{R}$, and let D be a database on $\mathcal{R}$. Then there exists a homomorphism μ such that $\mu(\mathit{Ochase}(D,\Sigma)) \subseteq \mathit{Rchase}(D,\Sigma)$.*

Proof. The proof is by induction on the number m of applications of the TGD chase rule in the construction of the oblivious chase $\mathit{Ochase}(D,\Sigma)$. We want to prove that, for all m with $m \geqslant 0$, there is a homomorphism from $\mathit{Ochase}^{[m]}(D,\Sigma)$ to $\mathit{Rchase}(D,\Sigma)$.
Base case. In the base case, where $m = 0$, no TGD rule has yet been applied, so $\mathit{Ochase}^{[0]}(D,\Sigma) = D \subseteq \mathit{Rchase}(D,\Sigma)$ and the required homomorphism is simply the identity homomorphism μ_0.
Inductive case. Assume we have applied the TGD chase rule m times and obtained $\mathit{Ochase}^{[m]}(D,\Sigma)$. By the induction hypothesis, there exists a homomorphism μ_m that maps $\mathit{Ochase}^{[m]}(D,\Sigma)$ into $\mathit{Rchase}(D,\Sigma)$. Consider the $(m+1)$-th application of the TGD chase rule, for a TGD of the form $\Phi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X},\mathbf{Z})$. By definition of applicability of TGDs, there is a homomorphism λ_O that maps $\Phi(\mathbf{X},\mathbf{Y})$ to atoms of $\mathit{Ochase}(D,\Sigma)$ and that it can be suitably extended to another homomorphism, λ'_O, such that λ'_O maps each of the variables in $\mathbf{Z}$ to a fresh null in Δ_N not already present in $\mathit{Ochase}^{[m]}(D,\Sigma)$. As a result of the application of this TGD, all atoms in $\lambda'_O(\Psi(\mathbf{X},\mathbf{Z}))$ are added to $\mathit{Ochase}^{[m]}(D,\Sigma)$, thus obtaining $\mathit{Ochase}^{[m+1]}(D,\Sigma)$. Consider the homomorphism $\lambda_R = \mu_m \circ \lambda_O$, which maps $\Phi(\mathbf{X},\mathbf{Y})$ to atoms of $\mathit{Rchase}(D,\Sigma)$. Since $\mathit{Rchase}(D,\Sigma)$ satisfies all the dependencies in Σ (and so does $\mathit{Ochase}(D,\Sigma)$), there is an extension λ'_R of λ_R that maps $\Psi(\mathbf{X},\mathbf{Z})$ to tuples of $\mathit{Rchase}(D,\Sigma)$. Denoting $\mathbf{Z} = Z_1,\ldots,Z_k$, we now define $\mu_{m+1} = \mu_m \cup \{\lambda'_O(Z_i) \rightarrow \lambda'_R(Z_i)\}_{1 \leqslant i \leqslant k}$. To complete the proof, we now need to show that μ_{m+1} is indeed a homomorphism. The addition of $\lambda'_O(Z_i) \rightarrow \lambda'_R(Z_i)$, with $1 \leqslant i \leqslant k$, is compatible with μ_m because none of the $\lambda'_O(Z_i)$ appears in μ_m. Therefore μ_{m+1} is a well-defined mapping. Now, consider an atom $r(\mathbf{X},\mathbf{Z})$ in $\Psi(\mathbf{X},\mathbf{Z})$. Then the atom $\lambda'_O(r(\mathbf{X},\mathbf{Z}))$ is added to $\mathit{Ochase}(D,\Sigma)$ in the $(m+1)$-th step and $\mu_{m+1}(r(\mathbf{X},\mathbf{Y})) = \mu_{m+1}(r(\lambda'_O(\mathbf{X}),\lambda'_O(\mathbf{Z}))) = r(\mu_{m+1}(\lambda'_O(\mathbf{X}),\mu_{m+1}(\lambda'_O(\mathbf{Z}))$. Notice that $\mu_{m+1}(\lambda'_O(\mathbf{X})) = \mu_{m+1}(\lambda_O(\mathbf{X})) = \lambda_R(\mathbf{X}) = \lambda'_R(\mathbf{X})$, and $\mu_{m+1}(\lambda'_O(\mathbf{Z})) = \lambda'_R(\mathbf{Z})$. Therefore, $\mu_{m+1}(r(\mathbf{X},\mathbf{Z})) = r(\lambda'_R(\mathbf{X}),\lambda'_R(\mathbf{Z})) = \lambda'_R(r(\mathbf{X},\mathbf{Z}))$, which is in $\mathit{Rchase}(D,\Sigma)$, by construction. The desired homomorphism from $\mathit{Ochase}(D,\Sigma)$ to $\mathit{Rchase}(D,\Sigma)$ is therefore $\mu = \bigcup_{i=0}^{\infty} \mu_i$. □

Corollary 2.14. *Given a set Σ of TGDs over a relational schema $\mathcal{R}$ and a database D for $\mathcal{R}$, $\mathit{Ochase}(D,\Sigma)$ is a universal solution for D under Σ.*

Corollary 2.15. *Given a Boolean query Q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set of TGDs Σ, $Ochase(D, \Sigma) \models Q$ if and only if $Rchase(D, \Sigma) \models Q$.*

In the following, unless explicitly stated otherwise, "chase" will mean the oblivious chase, and $chase(D, \Sigma)$ will stand for $Ochase(D, \Sigma)$.

2.8 Decision Problems

Recall that, by Theorem 2.10, $D \cup \Sigma \models Q$ iff $chase(D, \Sigma) \models Q$. Based on this, we define two relevant decision problems and prove their LOGSPACE-equivalence.

Definition 2.16. The *conjunctive query evaluation decision problem* CQeval is defined as follows. Given a conjunctive query Q with n-ary head predicate q, a set of TGDs Σ, a database D and a ground n-tuple $\mathbf{t}$, decide whether $\mathbf{t} \in ans(Q, D, \Sigma)$ or, equivalently, whether $chase(D, \Sigma) \cup \{Q\} \models q(\mathbf{t})$.

Definition 2.17. The *Boolean conjunctive query evaluation problem* BCQeval is defined as follows. Given a Boolean conjunctive query Q, a set of TGDs Σ, and a database D, decide whether $chase(D, \Sigma) \models Q$.

The following result is implicit in the work of Chandra and Merlin (1977).

Lemma 2.18. *The problems CQeval and BCQeval are* LOGSPACE*-equivalent.*

Proof. Notice that BCQeval can be trivially made into a special instance of CQeval, e.g., by adding a propositional atom as head atom. It thus suffices to show that CQeval polynomially reduces to BCQeval. Let $\langle Q, D, \Sigma, q(\mathbf{t}) \rangle$ be an instance of CQeval, where q/n is the head predicate of Q and $\mathbf{t}$ is a ground n-tuple. Assume the head atom of Q is $q(X_1, \ldots, X_n)$ and $\mathbf{t} = \langle c_1, \ldots, c_n \rangle$. Then define Q' to be the Boolean conjunctive query whose body is $body(Q) \wedge q'(X_1, \ldots, X_n)$, where q' is a fresh predicate symbol not occurring in D, Q, or Σ It is easy to see that $q(\mathbf{t}) \in Q(chase(D, \Sigma))$ iff $chase(D \cup \{q'(c_1, \ldots, c_n)\}, \Sigma) \models Q'$. $\square$

By the above lemma and by the well-known equivalence of the problem of query containment under TGDs with the CQeval problem (Corollary 2.12), the three following problems are LOGSPACE-equivalent: *(1)* CQ-eval under TGDs, *(2)* BCQeval under TGDs, *(3)* query containment under TGDs. Henceforth, we will consider only one of these problems, the BCQ-eval problem. By the above, all complexity results carry over to the other problems.

Dealing with multiple head-atoms. It turns out that dealing with multiple atoms in TGD heads complicates the proof techniques, so we assume that all TGDs have a single atom in their head. After proving our results for single-headed TGDs, we will extend these results to the case of multiple-atom heads in Section 8.

2.9 Tree Decomposition and Related Notions

We now introduce the required notions about tree decompositions. A *hypergraph* is a pair $\mathcal{H} = \langle V, H \rangle$, where V is the set of nodes and $H \subseteq 2^V$. The elements of H are thus subsets of V; they are called *hyperedges*. The *Gaifman graph* of a hypergraph $\mathcal{H} = \langle V, H \rangle$, denoted by $\mathcal{G}_{\mathcal{H}}$, is an undirected graph where V is the set of nodes and an edge (v_1, v_2) is in the graph if v_1 and v_2 jointly occur in some hyperedge in H.

Given a graph $\mathcal{G} = \langle V, E\rangle$, a *tree decomposition* of $\mathcal{G}$ is a pair $\langle T, \lambda\rangle$, where $T = \langle N, A\rangle$ is a tree, and λ a labeling function $\lambda : N \to 2^V$ such that:

(i) for all $v \in V$ there is $n \in N$ such that $v \in \lambda(n)$; that is, $\lambda(N) = \bigcup_{n\in N} \lambda(n) = V$;
(ii) for every edge $e = (v_1, v_2) \in E$ there is $n \in N$ such that $\lambda(n) \supseteq \{v_1, v_2\}$;
(iii) for every $v \in V$, the set $\{n \in N \mid v \in \lambda(n)\}$ induces a connected subtree in T.

The *width* of a tree decomposition $\langle T, \lambda\rangle$ is the integer value $\max\{|\lambda(n)| - 1 \mid n \in N\}$. The *treewidth* of a graph $\mathcal{G} = \langle V, E\rangle$, denoted by $\mathrm{tw}(\mathcal{G})$, is the minimum width of all tree decompositions of G. Given a hypergraph $\mathcal{H}$, its treewidth $\mathrm{tw}(\mathcal{H})$ is defined as the treewidth of its Gaifman graph: $\mathrm{tw}(\mathcal{H}) = \mathrm{tw}(\mathcal{G}_{\mathcal{H}})$. Notice that the notion of treewidth immediately extends to relational structures.

3. Guarded and Weakly-Guarded TGDs: Decidability Issues

This section introduces *guarded TGDs (GTGDs)* and *weakly guarded sets of TGDs (WGTGDs)*, which enjoy several useful properties. In particular, we show that query answering under these TGDs is decidable.

Definition 3.1. Given a TGD σ of the form $\Phi(\mathbf{X}, \mathbf{Y}) \to \Psi(\mathbf{X}, \mathbf{Z})$, we say that σ is *a (fully) guarded TGD (GTGD)* if there exists an atom in the body, called a *guard*, that contains all the universally quantified variables of σ, i.e., all the variables $\mathbf{X}, \mathbf{Y}$ that occur in $\Phi(\mathbf{X}, \mathbf{Y})$.

To define *weakly guarded* sets of TGDs, we first give the notion of an *affected* position in a predicate of a relational schema, given a set of TGDs Σ. Intuitively, a position π is affected in a set of TGDs Σ if there exists a database D such that a labeled null appears in some atom of $chase(D, \Sigma)$ at position π. The importance of affected positions for our definitions is that no labeled null can appear in non-affected positions. We define this notion below.

Definition 3.2. Given a relational schema $\mathcal{R}$ and a set of TGDs Σ over $\mathcal{R}$, a position π of a predicate p of $\mathcal{R}$ is *affected* with respect to Σ if either:

- *(base case)* for some $\sigma \in \Sigma$, an existentially quantified variable appears in π in $head(\sigma)$, or
- *(inductive case)* for some $\sigma \in \Sigma$, the variable appearing at position π in $head(\sigma)$ also appears in $body(\sigma)$, and *only* at affected positions.

Example 3.3. Consider the following set of TGDs:

$$\begin{array}{lrcl} \sigma_1: & p_1(X,Y), p_2(X,Y) & \to & \exists Z\, p_2(Y,Z) \\ \sigma_2: & p_2(X,Y), p_2(W,X) & \to & p_1(Y,X) \end{array}$$

Notice that $p_2[2]$ is affected since Z is existentially quantified in σ_1. The variable Y in σ_1 appears in $p_2[2]$ (which is an affected position) and also in $p_1[2]$ (which is not an affected position). Therefore Y in σ_1 does *not* make the position $p_2[1]$ an affected one. Similarly, in σ_2, X appears in the affected position $p_2[2]$ and also in the non-affected position $p_2[1]$. Therefore, $p_1[2]$ is not affected. On the other hand, Y in σ_2 appears in $p_2[2]$ and nowhere else. Since we have already established that $p_2[2]$ is an affected position, this makes $p_1[1]$ also an affected position. □

Definition 3.4. Consider a set of TGDs Σ on a schema $\mathcal{R}$. A TGD $\sigma \in \Sigma$ of the form $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists\, \Psi(\mathbf{X}, \mathbf{Z})$ is said to be *weakly guarded with respect to* Σ (*WGTGD*) if there is an atom in *body*(σ), called a *weak guard*, that contains all the universally quantified variables of σ that appear in affected positions with respect to Σ and do not also appear in non-affected positions with respect to Σ. The set Σ is said to be a *weakly guarded set of TGDs* if each TGD $\sigma \in \Sigma$ is weakly guarded with respect to Σ.

A GTGD or WGTGD may have more than one guard. In such a case, we will pick a lexicographically first guard or use some other criterion for fixing *the* guard of a rule. The actual choice will not affect our proofs and results.

The following theorem shows the undecidability of conjunctive query answering under TGDs. This result, in its general form, follows from undecidability results for TGD implication (see Beeri & Vardi, 1981; Chandra, Lewis, & Makowsky, 1981b). We show here that the CQ answering problem remains undecidable even in case of a *fixed* set Σ of single-headed TGDs with a *single* non-guarded rule, and a ground atom as query. Our proof is "from first principles" as it reduces the well-known halting problem for Turing machines to query-answering under TGDs. More recently, Baget, Leclère, Mugnier, and Salvat (2011a) showed that CQ answering is undecidable also in case Σ contains a single TGD, which, however, contains multiple atoms in its head.

Theorem 3.5. *There exists a fixed atomic BCQ Q and a fixed set of TGDs Σ_u, where all TGDs in Σ_u are guarded except one, such that it is undecidable to determine whether for a database D, $D \cup \Sigma_u \models Q$ or, equivalently, whether $chase(D, \Sigma_u) \models Q$.*

Proof. The proof hinges on the observation that, with appropriate input facts D, using a fixed set of TGDs that consists of guarded TGDs and a single unguarded TGD, it is possible to force an infinite grid to appear in $chase(D, \Sigma_u)$. By a further set of guarded rules, one can then easily simulate a deterministic universal Turing machine (TM) $\mathcal{M}$, which executes every deterministic TM with an empty input tape, whose transition table is specified in the database D. This is done by using the infinite grid, where the i-th horizontal line of the grid represents the tape content at instant i. We assume that transitions of the Turing machine $\mathcal{M}$ are encoded into a relation *trans* of D, where for example, the ground atom $trans(s_1, a_1, s_2, a_2, right)$ means *"if the current state is s_1 and symbol a_1 is read, then switch to state s_2, write a_2, and move to the right"*.

We show how the infinite grid is defined. Let D contain (among other initialization atoms that specify the initial configuration of $\mathcal{M}$) the atom $index(0)$, which defines the initial point of the grid. Also, we make use of three constants $right, left, stay$ for encoding the three types of moves. Consider the following TGDs:

$$index(X) \rightarrow \exists Y\ next(X, Y)$$

$$next(X, Y) \rightarrow index(Y)$$

$$trans(\mathbf{T}), next(X_1, X_2), next(Y_1, Y_2) \rightarrow grid(\mathbf{T}, X_1, Y_1, X_2, Y_2)$$

where $\mathbf{T}$ stands for the sequence of argument variables S_1, A_1, S_2, A_2, M, as appropriate for the predicate *trans*. Note that only the last of these three TGDs is non-guarded. The above TGDs define an infinite grid whose points have co-ordinates X and Y (horizontal and vertical, respectively) and where for each point its horizontal and vertical successors are also encoded. In addition, each point appears together with each possible transition rule. It is not hard to see that we can simulate the progress of our Turing machine $\mathcal{M}$ using suitable initialization atoms in D and guarded TGDs. To this end, we need additional predicates $\mathit{cursor}(Y, X)$, meaning that the cursor is in position X at time Y, $\mathit{state}(Y, S)$, expressing that $\mathcal{M}$ is in state S at time Y, and $\mathit{content}(X, Y, A)$, expressing that at time Y, the content of position X in the tape is A. The following rule encodes the behavior of $\mathcal{M}$ on all transition rules that move the cursor to the right:

$$
\begin{aligned}
&\mathit{grid}(S_1, A_1, S_2, A_2, \mathit{right}, X_1, Y_1, X_2, Y_2),\\
&\qquad\qquad \mathit{cursor}(Y_1, X_1), \mathit{state}(Y_1, S_1), \mathit{content}(X_1, Y_1, A_1) \rightarrow\\
&\qquad\qquad\qquad \mathit{cursor}(Y_2, X_2), \mathit{content}(X_1, Y_2, A_2), \mathit{state}(Y_2, S_2), \mathit{mark}(Y_1, X_1)
\end{aligned}
$$

Such a rule has also obvious sibling rules for “left” and “stay” moves. For the sake of brevity only, the above rule contains multiple atoms in the head. This is not a problem, as such rules have no existentially quantified variables in the head. Therefore, each TGD with multiple head-atoms can be replaced by an equivalent set of TGDs with single-atom heads and identical bodies.

Notice that the *mark* predicate in the head marks the tape cell that is modified at instant Y_1. We now need additional “inertia” rules, which ensure that all other positions in the tape are not modified between Y_1 and the following time instant Y_2. To this end, we use two different markings: keep_f for the tape positions that follow the one marked with *mark*, and keep_p for the preceding tape positions. In this way, we are able, by making use of guarded rules only, to ensure that, at every instant Y_1, every tape cell X, such that $\mathit{keep}_p(Y_1, X)$ or $\mathit{keep}_f(Y_1, X)$ is true, keeps the same symbol at the instant Y_2 following Y_1. The rules below then propagate the aforementioned markings forward and backwards, respectively, starting from the marked tape positions.

$$
\begin{array}{c}
\mathit{mark}(Y_1, X_1), \mathit{grid}(\mathbf{T}, X_1, Y_1, X_2, Y_2) \rightarrow \mathit{keep}_f(Y_1, X_2)\\
\mathit{keep}_f(Y_1, X_1), \mathit{grid}(\mathbf{T}, X_1, Y_1, X_2, Y_2) \rightarrow \mathit{keep}_f(Y_1, X_2)\\
\mathit{mark}(Y_1, X_2), \mathit{grid}(\mathbf{T}, X_1, Y_1, X_2, Y_2) \rightarrow \mathit{keep}_p(Y_1, X_1)\\
\mathit{keep}_p(Y_1, X_2), \mathit{grid}(\mathbf{T}, X_1, Y_1, X_2, Y_2) \rightarrow \mathit{keep}_p(Y_1, X_1)
\end{array}
$$

We also have inertia rules for all $a \in \{a_1, \ldots, a_\ell, \flat\}$, where $\{a_1, \ldots, a_\ell, \flat\}$ is the tape alphabet:

$$
\begin{array}{c}
\mathit{keep}_f(Y_1, X_1), \mathit{grid}(\mathbf{T}, X_1, Y_1, X_2, Y_2), \mathit{content}(X_1, Y_1, a) \rightarrow \mathit{content}(X_1, Y_2, a)\\
\mathit{keep}_p(Y_1, X_1), \mathit{grid}(\mathbf{T}, X_1, Y_1, X_2, Y_2), \mathit{content}(X_1, Y_1, a) \rightarrow \mathit{content}(X_1, Y_2, a)
\end{array}
$$

Notice that we use the constant a instead of a variable in the above rules in order to have the guardedness property. We therefore need two rules as above for every tape symbol, that is, $2\ell + 2$ inertia rules altogether.

Finally, we assume, without loss of generality, that our Turing machine $\mathcal{M}$ has a single halting state s_0 which is encoded by the atom $\mathit{halt}(s_0)$ in D. We then add a guarded rule $state(Y, S), halt(S) \rightarrow stop$. It is now clear that the machine halts iff $chase(D, \Sigma_u) \models stop$, i.e., iff $D \cup \Sigma_u \models stop$. We have thus reduced the halting problem to the problem of answering atomic queries over a database under Σ_u. The latter problem is therefore undecidable. $\square$

Definition 3.6. [Guarded chase forest, restricted GCF] Given a set of WGTGDs Σ and a database D, the *guarded chase forest (GCF)* for D and Σ, denoted $\mathrm{gcf}(D, \Sigma)$, is constructed as follows.

(a) For each atom (fact) $\underline{d}$ in D, add a node labeled with $\underline{d}$.

(b) For every node v labeled with $\underline{a} \in chase(D, \Sigma)$ and for every atom $\underline{b}$ obtained from $\underline{a}$ (and possibly other atoms) by a one-step application of a TGD $\sigma \in \Sigma$, if $\underline{a}$ is the image of the guard of σ then add one node v' labeled with $\underline{b}$ and an arc going from v to v'.

Assuming the chase forest $\mathrm{gcf}(D, \Sigma)$ is built inductively, following precisely the strategy of a fixed deterministic chase procedure, the set of all non-root nodes of the chase forest is totally ordered by a relation $\prec$ that reflects their order of generation. The *restricted GCF* for D and Σ, denoted $\mathrm{rgcf}(D, \Sigma)$, is obtained from $\mathrm{gcf}(D, \Sigma)$ by eliminating each subtree rooted in a node w whose label is a duplicate of an earlier generated node. Thus, if v and w are nodes labeled by the same atom, and $v \prec w$, w and all nodes of the subtree rooted in w are eliminated from $\mathrm{gcf}(D, \Sigma)$ so as to obtain $\mathrm{rgcf}(D, \Sigma)$. Note that in $\mathrm{rgcf}(D, \Sigma)$ each label occurs only once, therefore we can identify the nodes with their labels and say, for instance, "the node $\underline{a}$" instead of "the node v labeled by $\underline{a}$".

Example 3.7. Consider again Example 2.9 on page 123. The corresponding (infinite) guarded chase forest is shown in Figure 2. Every edge from an $\underline{a}$-node to a $\underline{b}$-node is labeled with the TGD whose application causes the introduction of $\underline{b}$. Notice that some atoms (e.g., $r_2(b)$ or $r_2(z_2)$) label more than one node in the forest. The nodes belonging also to the restricted GCF are shaded in the figure. $\square$

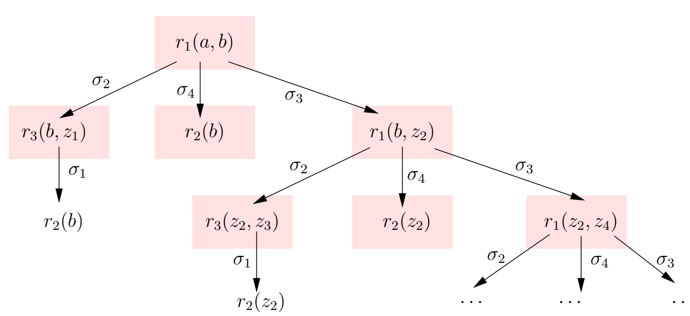

Figure 2: Chase forest for Example 3.7.

The goal of the following material is to show that, for weakly guarded sets Σ of TGDs, the possibly infinite set of atoms $chase(D, \Sigma)$ has finite treewidth (Lemma 3.13). This will then be used to show the decidability of query-answering under WGTGDs (Theorem 3.14). As a first step towards proving that $chase(D, \Sigma)$ has finite treewidth, we generalize the notion of acyclicity of an instance, and then point out the relationship between this notion and treewidth. We will then show that $chase(D, \Sigma)$ enjoys (a specific version of) our generalized form of acyclicity (Lemma 3.11), from which the finite treewidth result immediately follows.

Definition 3.8. Let B be a (possibly infinite) instance for a schema $\mathcal{R}$ and let $S \subseteq dom(B)$.

- An $[S]$-join forest $\langle F, \mu \rangle$ of B is an undirected labeled forest $F = \langle V, E \rangle$ (finite or infinite), whose labeling function $\mu : V \to B$ is such that:
  (1) μ is an epimorphism, i.e., $\mu(V) = B$;
  (2) F is $[S]$-connected, i.e., for each $c \in dom(B) - S$, the set $\{v \in V \mid c \in dom(\mu(v))\}$ induces a connected subtree in F.
- We say that B is $[S]$-*acyclic* if B has an $[S]$-join forest.

Notice that we are dealing with a relational instance, but the above definition works for any relational structure, including queries. Definition 3.8 generalizes the classical notion of hypergraph acyclicity (Beeri, Fagin, Maier, Mendelzon, Ullman, & Yannakakis, 1981) of an instance or of a query: an instance or a query, seen as a hypergraph, is hypergraph-acyclic (which is the same as α-acyclic according to Fagin, 1983) if and only if it is $[\varnothing]$-acyclic .

The following Lemma follows from the definitions of $[S]$-acyclicity.

Lemma 3.9. *Given an instance B for a schema $\mathcal{R}$, and a set $S \subseteq dom(B)$, if B is $[S]$-acyclic, then* $\mathrm{tw}(B) \leqslant |S| + w$, *where w is the maximum predicate arity in $\mathcal{R}$ and* $\mathrm{tw}(B)$ *is the treewidth of B.*

Proof. By hypothesis, B is $[S]$-acyclic and therefore has an $[S]$-join forest $\langle F, \mu \rangle$, where $F = \langle V, E \rangle$. A tree decomposition $\langle T, \lambda \rangle$ with $T = \langle N, A \rangle$, is constructed as follows. First, we take $N = V \cup \{n_0\}$, where n_0 is an auxiliary node. Let $V_r \subseteq V$ be the set of nodes that are roots in the $[S]$-join forest F and let A_r be the set of edges from n_0 to each node in V_r. We define $A = E \cup A_r$. The labeling function is defined as follows: $\lambda(n_0) = S$, and for all nodes $v \neq n_0$, $\lambda(v) = dom(\mu(v)) \cup S$. We now show that $\langle T, \lambda \rangle$ is a tree decomposition. Recalling the definition of tree decompositions in Section 2.9, *(i)* holds trivially because F is a join forest and $\mu(V) = B$. As for *(ii)*, we notice that edges in the Gaifman graph of B are such that for each atom $\underline{d} = r(c_1, \ldots, c_m)$ in B there is a clique among nodes $c_1, \ldots, c_m$. Since for the same atom there exists $v \in V$ such that $\mu(v) = \underline{d}$ and $\lambda(v) \supseteq dom(\mu(v))$, *(ii)* holds immediately. Finally we consider connectedness. Let us take a value c appearing in B as argument. If $c \in S$, the set $\{v \in N \mid c \in \lambda(v)\}$ is the entire N, by construction, so connectedness holds. If $c \notin S$, the set $\{v \in N \mid c \in \lambda(v)\}$ induces a connected subtree in F and therefore in T, since $\lambda(v) = \mu(v) \cup S$. Therefore, *(iii)* holds. Notice also that the width of such a tree decomposition is at most $|S| + w$ by construction. $\square$

Definition 3.10. Let D be a database for a schema $\mathcal{R}$, and $HB(D)$ be the Herbrand Base of D as defined in Section 2. We define:

- $chase^{\perp}(D, \Sigma) = chase(D, \Sigma) \cap HB(D)$, and
- $chase^{+}(D, \Sigma) = chase(D, \Sigma) - chase^{\perp}(D, \Sigma)$

In plain words, $chase^{\perp}(D, \Sigma)$ is the finite set of all null-free atoms in $chase(D, \Sigma)$. In contrast, $chase^{+}(D, \Sigma)$ may be infinite; it is the set of all atoms in $chase(D, \Sigma)$ that have at least one null as an argument. Note that $chase^{\perp}(D, \Sigma) \cup chase^{+}(D, \Sigma) = chase(D, \Sigma)$ and $chase^{\perp}(D, \Sigma) \cap chase^{+}(D, \Sigma) = \varnothing$.

Lemma 3.11. *If Σ is a weakly guarded set of TGDs and D a database, then $chase^{+}(D, \Sigma)$ is $[dom(D)]$-acyclic, and so is therefore $chase(D, \Sigma)$.*

In order to prove this result, we resort to an auxiliary lemma.

Lemma 3.12. *Let D be a database and Σ a weakly guarded set of TGDs. Let $\underline{a}_s$ be a node of* $\mathrm{rgcf}(D, \Sigma)$ *where the null value $\zeta \in \Delta_N$ is* first *introduced, and let $\underline{a}_f$ be a descendant node of $\underline{a}_s$ in* $\mathrm{rgcf}(D, \Sigma)$ *that has ζ as an argument. Then, ζ appears in every node (=atom) on the (unique) path from $\underline{a}_s$ to $\underline{a}_f$.*

Proof. Let $\underline{a}_1 = \underline{a}_s, \underline{a}_2, \ldots, \underline{a}_n = \underline{a}_f$ be the path from $\underline{a}_s$ to $\underline{a}_f$. By the definition of affected positions, ζ appears *only* in affected positions in the atoms in the chase. Suppose, to the contrary, that ζ does not appear in some intermediate atom in the above path. Then, there is i, $2 \leqslant i \leqslant n-1$, such that ζ does not appear in $\underline{a}_i$, but appears in $\underline{a}_{i+1}$. Since ζ appears only in affected positions, in order to be in $\underline{a}_{i+1}$ it must either appear in $\underline{a}_i$ or to be invented when $\underline{a}_{i+1}$ was added. The first case is ruled out by the assumption, and the second is impossible because ζ was first introduced in $\underline{a}_1$, not in $\underline{a}_{i+1}$—a contradiction. □

We now come back to the proof of Lemma 3.11.

Proof. The proof is constructive, by exhibiting a $[dom(D)]$-join forest $F = \langle V, E \rangle$ for $chase(D, \Sigma)$. We take F as $\mathrm{rgcf}(D, \Sigma)$ and define, for each atom $\underline{d} \in \mathrm{rgcf}(D, \Sigma)$, the labeling function μ for F as $\mu(\underline{d}) = \underline{d}$. Since every atom of $chase(D, \Sigma)$ is "covered" by its corresponding node of F, it only remains to show that $chase(D, \Sigma)$ is $[dom(D)]$-connected. Take a pair of distinct atoms $\underline{a}_1, \underline{a}_2$ in $\mathrm{rgcf}(D, \Sigma)$ that both have the same value $c \in \Delta_N$ in an argument. The atoms $\underline{a}_1$ and $\underline{a}_2$ must have a common ancestor $\underline{a}$ in $\mathrm{rgcf}(D, \Sigma)$ where c was first invented: if they do not, the value c would have to be introduced twice in $chase(D, \Sigma)$. By Lemma 3.12, c appears in all atoms on the paths from $\underline{a}$ to $\underline{a}_1$ and from $\underline{a}$ to $\underline{a}_2$. It thus follows that the set $\{v \in V \mid c \in \mu(v)\}$ induces a connected subtree in F. □

Lemma 3.13. *If Σ is a weakly guarded set of TGDs and D a database for a schema $\mathcal{R}$, then* $\mathrm{tw}(chase(D, \Sigma)) \leqslant |dom(D)| + w$*, where w is the maximum predicate arity in $\mathcal{R}$.*

Proof. The claim follows from Lemmas 3.9 and 3.11. □

Theorem 3.14. *Given a relational schema $\mathcal{R}$, a weakly guarded set of TGDs Σ, a Boolean conjunctive query Q, and a database D for $\mathcal{R}$, the problem of checking whether $D \cup \Sigma \models Q$, or equivalently $chase(D, \Sigma) \models Q$, is decidable.*

Proof (sketch). We first remind a key result of Courcelle (1990), that generalizes an earlier result of Rabin (1969). Courcelle's result states that the satisfiability problem is decidable for classes of first-order theories (more generally, theories of monadic second-order logic) that enjoy the finite treewidth model property. A class $\mathcal{C}$ of theories has the finite-treewidth model property if for each satisfiable theory $\mathcal{T} \in \mathcal{C}$ it is possible to compute an integer $f(\mathcal{T})$ such that $\mathcal{T}$ has a model of treewidth at most $f(\mathcal{T})$—see also the works by Goncalves and Grädel (2000) and by Grädel (1999), where a more general property, called the generalized tree-model property, is discussed. We apply this to prove our theorem.

Let $\neg Q$ be the universal sentence obtained by negating the existentially quantified conjunctive query Q. For all classes of TGDs, $D \cup \Sigma \models Q$ iff $\mathit{chase}(D, \Sigma) \models Q$ iff $D \cup \Sigma \cup \neg Q$ is unsatisfiable. Trivially, deciding whether $D \cup \Sigma \models Q$ is equivalent under Turing reductions to deciding whether $D \cup \Sigma \not\models Q$. The latter holds iff $D \cup \Sigma \cup \{\neg Q\}$ is satisfiable or, equivalently, iff $\mathit{chase}(D, \Sigma)$ is a model of $\neg Q$ which, in turn, holds iff $\mathit{chase}(D, \Sigma)$ is a model of $D \cup \Sigma \cup \{\neg Q\}$. By Lemma 3.13, for WGTGDs, $\mathit{chase}(D, \Sigma)$ has finite treewidth. Our decision problem thus amounts to checking whether a theory belonging to a class $\mathcal{C}^*$ of first-order theories (of the form $D \cup \Sigma \cup \{\neg Q\}$) is satisfiable, where it is guaranteed that whenever a theory in this class is satisfiable, then it has a model of finite treewidth (namely, $\mathit{chase}(D, \Sigma)$), and where $\mathcal{C}^*$ therefore enjoys the finite treewidth model property. Decidability thus follows from Courcelle's result. □

Determining the precise complexity of query answering under sets of guarded and weakly guarded sets of TGDs will require new techniques, which are the subject of the next sections.

4. Complexity: Lower Bounds

In this section we prove several lower bounds for the complexity of the decision problem of answering Boolean conjunctive queries under guarded and weakly guarded sets of TGDs.

Theorem 4.1. *The problem* BCQeval *under WGTGDs is* EXPTIME*-hard in case the TGDs are fixed. The same problem is* 2EXPTIME*-hard when the predicate arity is not bounded. Both hardness results also hold for fixed atomic ground queries.*

Proof. We start with the EXPTIME-hardness result for fixed WGTGD sets Σ. It is well-known that APSPACE (alternating PSPACE, see Chandra, Kozen, & Stockmeyer, 1981a) equals EXPTIME. Notice that there are APSPACE-hard languages that are accepted by alternating polynomial-space machines that use at most n worktape cells, where n is the input size, and where the input is initially placed on the worktape. (This is well-known and can be shown by trivial padding arguments). To prove our claim, it thus suffices to simulate the behavior of such a restricted linear space (LINSPACE) Alternating Turing Machine (ATM) $\mathcal{M}$ on an input bit string I by means of a weakly guarded set of TGDs Σ and a database D. Actually, we will show a stronger result: that a *fixed* set Σ of WGTGDs can simulate a *universal* ATM that in turn simulates every LINSPACE ATM that uses at most n tape cells on every input. Here both the ATM transition table and the ATM input string will be stored in the database D. Then $D \cup \Sigma \models Q$ for some atomic ground query Q iff the ATM accepts the given input.[2]

Without loss of generality, we can assume that the ATM $\mathcal{M}$ has exactly one accepting state s_a. We also assume that $\mathcal{M}$ never tries to read beyond its tape boundaries. Let $\mathcal{M}$

2. This technique was proposed by Calì et al. (2008). It is similar to a technique later described by Hernich, Libkin, and Schweikardt (2011) in the proof of undecidability of the existence of so-called CWA (closed-world assumption) universal solutions in data exchange.

be defined as

$$\mathcal{M} = (S, \Lambda, \flat, \delta, s_0, \{s_a\})$$

where S is the set of states, $\Lambda = \{0, 1, \flat\}$ is the tape alphabet, $\flat$ is the blank tape symbol, δ is the transition function, defined as $\delta : S \times \Lambda \to (S \times \Lambda \times \{\ell, r, \bot\})^2$ ($\bot$ denotes the "stay" head move, while ℓ and r denote "left" and "right", respectively), $s_0 \in S$ is the initial state, and $\{s_a\}$ is the singleton-set of accepting states. Since $\mathcal{M}$ is an alternating Turing machine (ATM), its set of states S is *partitioned* into two sets: $S_\forall$ and $S_\exists$ (universal and existential states, respectively). The general idea of the encoding is that configurations of $\mathcal{M}$ (except for the initial configuration κ) will be represented by fresh nulls v_i, $i \geqslant 1$, that are generated by the chase.

The relational schema. We now describe the predicates of the schema which we use in the reduction. Notice that the schema is *fixed* and does not depend on the particular ATM that we encode. The schema predicates are as follows.

(1) Tape. The ternary predicate $symbol(a, c, v)$ denotes that in configuration v the cell c contains the symbol a, with $a \in \Lambda$. Also, a binary predicate $succ(c_1, c_2)$ denotes the fact that cell c_1 *follows* cell c_2 on the tape. Finally, $neq(c_1, c_2)$ says that two cells are distinct.

(2) States. A binary predicate $state(s, v)$ says that in configuration v the ATM $\mathcal{M}$ is in state s. We use three additional unary predicates: *existential*, *universal*, and *accept*. The atom $existential(s)$ (resp., $universal(s)$) denotes that the state s is existential (resp., universal), while $accept(c)$ says that c is an accepting configuration, that is, one whose state is the accepting state.

(3) Configurations. A unary predicate $config(v)$ expresses the fact that the value v identifies a configuration. A ternary predicate $next(v, v_1, v_2)$ is used to say that both configurations v_1 and v_2 are derived from v. Similarly, we use $follows(v, v')$ to say that configuration v' is derived from v. Finally, a unary predicate $init(v)$ states that the configuration v is initial.

(4) Head (cursor). We use the fact $cursor(c, v)$ to say that the head (cursor) of the ATM is on cell c in configuration v.

(5) Marking. Similarly to what is done in the proof of Theorem 3.5, we use $mark(c, v)$ to say that a cell c is marked in a configuration v. Our TGDs will ensure that all non-marked cells keep their symbols in a transition from one configuration to another.

(6) Transition function. To represent the transition function δ of $\mathcal{M}$, we use a single 8-ary predicate *transition*: for every transition rule $\delta(s, a) = ((s_1, a_1, m_1), (s_2, a_2, m_2))$ we will have $transition(s, a, s_1, a_1, m_1, s_2, a_2, m_2)$.

The database D. The data constants of the database D are used to identify cells, configurations, states and so on. In particular, we will use an accepting state s_a and an initial state s_0 plus a special initial configuration κ. The database describes the initial configuration of the ATM with some technicalities.

(a) We assume, without loss of generality, the n symbols of the input I to occupy the cells numbered from 1 to n, i.e., $c_1, \ldots, c_n$. For technical reasons, in order to obtain a simpler TGD set below, we also use the dummy cell constants c_0 and c_{n+1}, that intuitively represent border cells without symbols. For the i-th symbol a_i of I, the database has the fact $symbol(a, c_i, \kappa)$, for all $i \in \{1, \ldots, n\}$.

(b) An atom $\mathit{state}(s_0, \kappa)$ specifies that $\mathcal{M}$ is in state s_0 in its initial configuration κ.
(c) For every existential state s_E and universal state s_U, we have the facts $\mathit{existential}(s_E)$ and $\mathit{universal}(s_U)$. For the accepting state, the database has the fact $\mathit{accept}(s_a)$.
(d) An atom $\mathit{cursor}(c_1, \kappa)$ indicates that, in the initial configuration, the cursor points at the first cell.
(e) The atoms $\mathit{succ}(c_1, c_2), \ldots, \mathit{succ}(c_{n-1}, c_n)$ encode the fact that the cells $c_1, \ldots, c_n$ of the tape (beyond which the ATM does not operate) are adjacent. For technical reasons, we also use the analogous facts $\mathit{succ}(c_0, c_1)$ and $\mathit{succ}(c_n, c_{n+1})$. Also, atoms of the form $\mathit{neq}(c_i, c_j)$, for all i, j such that $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n$ and $i \neq j$, denote the fact that the cells $c_1, \ldots, c_n$ are pairwise distinct.
(f) The atom $\mathit{config}(\kappa)$ says that κ is a valid configuration.
(g) The database has atoms of the form $\mathit{transition}(s, a, s_1, a_1, m_1, s_2, a_2, m_2)$, which encode the transition function δ, as described above.

The TGDs. We now describe the TGDs that define the transitions and the accepting configurations of the ATM.

(a) Configuration generation. The following TGDs say that, for every configuration (halting or non halting—we do not mind having configurations that are derived from a halting one), there are two configurations that follow it, and that a configuration that follows another configurations is also a valid configuration:

$$\begin{array}{rcl} \mathit{config}(V), & \rightarrow & \exists V_1 \exists V_2\, \mathit{next}(V, V_1, V_2) \\ \mathit{next}(V, V_1, V_2) & \rightarrow & \mathit{config}(V_1), \mathit{config}(V_2) \\ \mathit{next}(V, V_1, V_2) & \rightarrow & \mathit{follows}(V, V_1) \\ \mathit{next}(V, V_1, V_2) & \rightarrow & \mathit{follows}(V, V_2) \end{array}$$

(b) Configuration transition. The following TGD encodes the transition where the ATM starts at an existential state, moves right in its first configuration and left in the second. Here C denotes the current cell, C_1 and C_2 are the new cells in the first and the second configuration (on the right and on the left of C, respectively), and the constants r, ℓ, and $\perp$ represent the "right," the "left," and the "stay" moves, respectively.

$$\begin{array}{l} \mathit{transition}(S, A, S_1, A_1, r, S_2, A_2, \ell), \mathit{next}(V, V_1, V_2), \\ \quad \mathit{state}(S, V), \mathit{cursor}(C, V), \mathit{symbol}(A, C, V), \mathit{succ}(C_1, C), \mathit{succ}(C, C_2) \rightarrow \\ \quad\quad \mathit{state}(S_1, V_1), \mathit{state}(S_2, V_2), \mathit{symbol}(A_1, C_1, V_1), \mathit{symbol}(A_2, C_2, V_2), \\ \quad\quad\quad \mathit{cursor}(C_1, V_1), \mathit{cursor}(C_2, V_2), \mathit{mark}(C, V), \end{array}$$

There are nine rules like the above one, corresponding to all the possible moves of the head in the child configurations C_1 and C_2. These other moves are encoded via similar TGDs. These rules suitably mark the cells that are written by the transition by means of the predicate *mark*. The cells that are not involved in the transition must retain their symbols, which is specified by the following TGD:

$$\mathit{config}(V), \mathit{follows}(V, V_1), \mathit{mark}(C, V), \mathit{symbol}(C_1, A, V), \mathit{neq}(C_1, C) \rightarrow \mathit{symbol}(C_1, A, V_1)$$

(c) Termination. The rule $state(s_a, V) \rightarrow accept(V)$ defines a configuration V to be accepting if its state is the accepting state. The following TGDs state that, for an existential state, at least one configuration derived from it must be accepting. For universal states, both configurations must be accepting.

$$next(V, V_1, V_2), state(S, V), existential(S), accept(V_1) \rightarrow accept(V)$$

$$next(V, V_1, V_2), state(S, V), existential(S), accept(V_2) \rightarrow accept(V)$$

$$next(V, V_1, V_2), state(S, V), universal(S), accept(V_1), accept(V_2) \rightarrow accept(V)$$

Note that, for brevity, some of the TGDs we used have multiple atoms in the head. However, these heads have no existentially quantified variables, so such multi-headed TGDs can be replaced with sets of TGDs that have only one head-atom. Note also that the database constants (r, ℓ, and $\perp$, and s_a) appearing in some rules can be eliminated by introducing additional predicate symbols and database atoms. For example, if we add the predicate *acceptstate* to the signature and the fact $acceptstate(s_a)$ to the database D, the rule $state(s_a, V) \rightarrow accept(V)$ can be replaced by the equivalent constant-free rule $acceptstate(X), state(X, V) \rightarrow accept(V)$.

It is not hard to show that the encoding described above is sound and complete. That is, $\mathcal{M}$ accepts the input I if and only if $chase(D, \Sigma) \models accept(\kappa)$. It is also easy to verify that the set of TGDs we have used is weakly guarded—this can be done by checking that each variable appearing only in affected positions also appears in a guard atom. For instance, take the above rule $next(V, V_1, V_2), state(S, V), existential(S), accept(V_1) \rightarrow accept(V)$. It is immediate to see that $state[1]$ and $existential[1]$ are non-affected (the TGDs never invent new states), and that all variables appearing in affected positions only, namely V, V_1, V_2, appear in the guard atom $next(V, V_1, V_2)$. This proves the claim.

We now turn to the case where Σ is not fixed and has unbounded predicate arities. For obtaining the 2EXPTIME lower bound, it is sufficient to adapt the above proof so as to simulate an ATM having 2^n worktape cells, i.e., an AEXPSPACE machine whose space is restricted to 2^n tape cells. Actually, to accommodate two dummy cells to the left and right of the 2^n effective tape cells, that are used for technical reasons, we will feature 2^{n+1} tape cells instead of just 2^n.

We will make sure that the input string is put on cells $1, \ldots, n$ of the worktape. Given that there are now many more than n worktape cells, we will fill all cells to the right of the input string with the blank symbol $\flat$.

This time, the WGTGD set Σ will not be fixed, but will depend on n. Since a much stronger result will be shown in Section 6 (Theorem 6.2), we do not belabor all the details in what follows, but just explain how the above proof for fixed sets of TGDs needs to be changed.

Rather than representing each tape cell by a data constant, each tape cell is now represented by a vector $(b_0, b_1, b_2, \ldots, b_n)$ of Boolean values from $\{0, 1\}$. The database D is the same as before, except for the following changes:

- D contains the additional facts $bool(0)$, $bool(1)$, $zero(0)$, $one(1)$.
- Each fact $symbol(a, c_i, \kappa)$ is replaced by the fact $symbol(a, b_0, b_1, b_2, \ldots, b_n, \kappa)$, where $(b_0, b_1, b_2, \ldots, b_n)$ is the Boolean vector of length n representing the integer i, with $0 \leqslant i \leqslant n \leqslant 2^{n+1}$.

- The fact $cursor(c_1, \kappa)$ is replaced by the $(n+2)$-ary fact $cursor(0, 0, \cdots, 0, 1, \kappa)$.
- All *succ* and *neq* facts described under item *(e)* are eliminated. (Vectorized versions of these predicates will be defined via Datalog rules—see below).

The TGD set from before is changed as follows. In all rules, each cell-variable C is replaced by a vector $\mathbf{C}$ of n variables. For example, the atom $succ(C_1, C)$ now becomes $succ(\mathbf{C}_1, \mathbf{C}) = succ(C_1^0, C_1^1, \ldots C_1^n, C^0, C^1, \ldots, C^n)$.

We add Datalog rules for n-ary *succ* and *neq* predicates. For example, the n-ary predicate *succ* can be implemented by the following rules:

$$
\begin{array}{rcl}
bool(X_0), \ldots, bool(X_{n-1}) & \rightarrow & succ(X_0, \ldots, X_{n-1}, 0, X_0, \ldots, X_{n-1}, 1), \\
bool(X_0), \ldots, bool(X_{n-2}) & \rightarrow & succ(X_0, \ldots, X_{n-2}, 0, 1, X_0, \ldots, X_{n-2}, 1, 0), \\
& \vdots & \\
bool(X_0), \ldots, bool(X_{n-i}) & \rightarrow & succ(X_0, \ldots, X_{n-i}, 0, 1 \ldots 1, X_0, \ldots, X_{n-i}, 1, 0, \ldots, 0), \\
& \vdots & \\
& \rightarrow & succ(0, 1, \ldots, 1, 1, 0, \ldots, 0)
\end{array}
$$

These rules contain constants which can be easily eliminated by use of the *zero* and *one* predicates, which are extensional database (EDB) predicates. We further add simple Datalog rules that use the vectorized *succ* predicate to define vectorized versions for the *less_than* and the *neq* predicates. Using *less_than*, we add a single rule that, for the initial configuration κ, puts blanks into all tape cells beyond the last cell n of the input: $less\_than(\mathbf{n}, \mathbf{C}) \rightarrow symbol(\flat, \mathbf{C}, \kappa)$, where $\mathbf{n}$ is an n-ary binary vector representing the number n (i.e., the input size).

The resulting set of rules is weakly guarded and correctly simulates the AEXPSPACE (alternating exponential space) Turing machine whose transition table is stored in the database D. Our reduction is polynomial in time. Since AEXPSPACE=2EXPTIME, it immediately follows that when the arity is not bounded the problem is 2EXPTIME-hard. □

5. Complexity: Upper Bounds

In this section we present upper bounds for query answering under weakly guarded TGDs.

5.1 Squid Decompositions

We now define the notion of a *squid decomposition*, and prove a lemma called "Squid Lemma" which will be a useful tool for proving our complexity results in the following sub-sections.

Definition 5.1. Let Q be a Boolean conjunctive query with n body atoms over a schema $\mathcal{R}$. An $\mathcal{R}$-*cover* of Q is a Boolean conjunctive query Q^+ over $\mathcal{R}$ that contains in its body all the body atoms of Q. In addition, Q^+ may contain at most n other $\mathcal{R}$-atoms.

Example 5.2. Let $\mathcal{R} = \{r/2, s/3, t/3\}$ and Q be the Boolean conjunctive query with body atoms $\{r(X,Y), r(Y,Z), t(Z,X,X)\}$. The following query Q^+ is an $\mathcal{R}$-cover of Q: $Q^+ = \{r(X,Y), r(Y,Z), t(Z,X,X), t(Y,Z,Z), s(Z,U,U)\}$. □

Lemma 5.3. *Let B be an instance over a schema $\mathcal{R}$ and Q a Boolean conjunctive query over B. Then $B \models Q$ iff there exists an $\mathcal{R}$-cover Q^+ of Q such that $B \models Q^+$.*

Proof. The *only-if* direction follows trivially from the fact that Q is an $\mathcal{R}$-cover of itself. The *if* direction follows straightforwardly from the fact that whenever there is a homomorphism $h : \mathit{vars}(Q^+) \to \mathit{dom}(B)$, such that $h(Q^+) \subseteq B$, then, given that Q is a subset of Q^+, the restriction h' of h to $\mathit{vars}(Q)$ is a homomorphism $\mathit{vars}(Q) \to \mathit{dom}(B)$ such that $h'(Q) = h(Q) \subseteq B$. Therefore $B \models Q^+$ implies $B \models Q$. $\square$

Definition 5.4. Let Q be a Boolean conjunctive query over a schema $\mathcal{R}$. A *squid decomposition* $\delta = (Q^+, h, H, T)$ of Q consists of an $\mathcal{R}$-cover Q^+ of Q, a mapping $h : \mathit{vars}(Q^+) \to \mathit{vars}(Q^+)$, and a partition of $h(Q^+)$ into two sets H and T, with $T = h(Q^+) - H$, for which there exists a set of variables $V_\delta \subseteq h(\mathit{vars}(Q^+))$ such that: *(i)* $H = \{\underline{a} \in h(Q^+) \mid \mathit{vars}(\underline{a}) \subseteq V_\delta\}$, and *(ii)* T is $[V_\delta]$-acyclic. If an appropriate set V_δ is given together with a squid decomposition $\delta = (Q^+, h, H, T)$, then, by a slight terminology overloading, we may just speak about *the squid decomposition* (Q^+, h, H, T, V_δ).

Note that a squid decomposition $\delta = (Q^+, h, H, T)$ of Q does not necessarily define a query folding (Chandra & Merlin, 1977; Qian, 1996) of Q^+, because h does not need to be an endomorphism of Q^+; in other terms, we do not require that $h(Q^+) \subseteq Q^+$. However, h is trivially a homomorphism from Q^+ to $h(Q^+)$.

Intuitively, a squid decomposition $\delta = (Q^+, h, H, T, V_\delta)$ describes a way how a query Q may be mapped homomorphically to $\mathit{chase}(D, \Sigma)$. First, instead of mapping Q to $\mathit{chase}(D, \Sigma)$, we equivalently map $h(Q^+) = H \cup T$ to $\mathit{chase}(D, \Sigma)$. The set V_δ specifies those variables of $h(Q^+)$ that ought to be mapped to constants, i.e., to elements of $\mathit{dom}(D)$. The atoms set H is thus mapped to ground atoms, that is, elements of the finite set $\mathit{chase}^\perp(D, \Sigma)$, which may be highly cyclic. The $[V_\delta]$-acyclic atom set T shall be mapped to the possibly infinite set $\mathit{chase}^+(D, \Sigma)$ which, however, is $[\mathit{dom}(D)]$-acyclic. The "acyclicities" of $\mathit{chase}^+(D, \Sigma)$ and of T will be exploited for designing appropriate decision procedures for determining whether $\mathit{chase}(D, \Sigma) \models Q$. All this will be made formal in the sequel.

One can think of the set H in a squid decomposition $\delta = (Q^+, h, H, T, V_\delta)$ as the *head* of a squid, and the set T as a join-forest of *tentacles* attached to that head. This will become clear in the following example and the associated Figure 3.

Example 5.5. Consider the following Boolean conjunctive query:

$$\begin{aligned}
&Q = \{r(X, Y), r(X, Z), r(Y, Z),\\
&r(Z, V_1), r(V_1, V_2), r(V_2, V_3), r(V_3, V_4), r(V_4, V_5),\\
&r(V_1, V_6), r(V_6, V_5), r(V_5, V_7), r(Z, U_1), s(U_1, U_2, U_3),\\
&r(U_3, U_4), r(U_3, U_5), r(U_4, U_5)\}.
\end{aligned}$$

Let Q^+ be the following Boolean query: $Q^+ = Q \cup \{s(U_3, U_4, U_5)\}$. A possible squid decomposition (Q^+, h, H, T, V_δ) can be based on the homomorphism h, defined as follows: $h(V_6) = V_2$, $h(V_4) = h(V_5) = h(V_7) = V_3$, and $h(X) = X$ for any other variable X in Q^+. The result of the squid decomposition with $V_\delta = \{X, Y, Z\}$ is the query shown in Figure 3.

Here the cyclic head H (encircled in the oval) is represented by its *join graph*,[3] and the $[V_\delta]$-acyclic "tentacle" set T is depicted as a $[V_\delta]$-join forest. Moreover, the forest representing T is rooted in the "bag" of H-atoms, so that the entire decomposition takes on the shape of a squid. Note that if we eliminated the additional atom $s(U_3, U_4, U_5)$, the original set of atoms $\{r(U_3, U_4), r(U_3, U_5), r(U_4, U_5)\}$ would form a non-$[V_\delta]$-acyclic cycle, and therefore they would not all be part of the tentacles. □

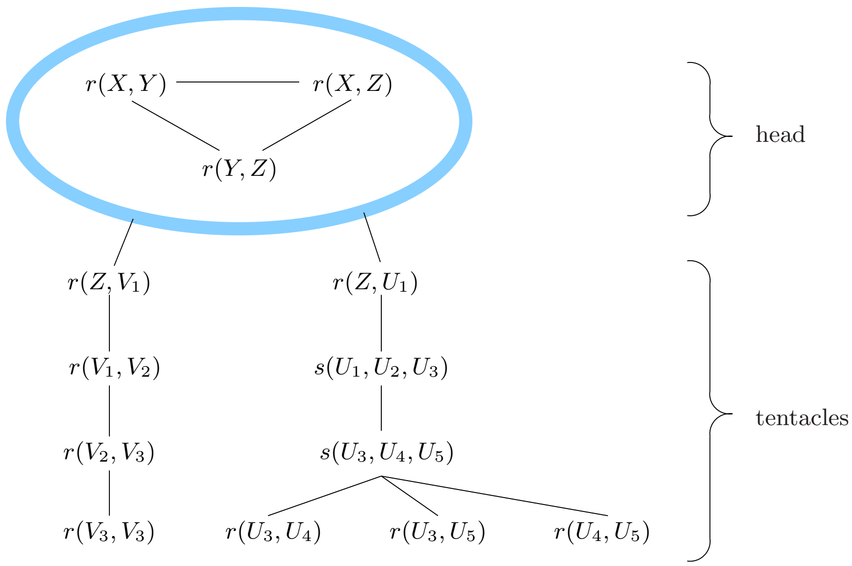

Figure 3: Squid decomposition from Example 5.5. Atoms in $h(Q^+)$ are shown.

The following two lemmas are auxiliary technical results.

Lemma 5.6. *Let Q be a Boolean conjunctive query and let U be a (possibly infinite) $[A]$-acyclic instance, where $A \subseteq dom(U)$. Assume $U \models Q$, i.e., there is a homomorphism $f : dom(Q) \to dom(U)$ with $f(Q) \subseteq U$. Then:*

(1) There is an $[A]$-acyclic subset $W \subseteq U$ such that: (i) $f(Q) \subseteq W$ and (ii) $|W| < 2|Q|$.

(2) There is a cover Q^+ of Q such that $|Q^+| < 2|Q|$, and there is a homomorphism g that extends f and $g(Q^+) = W$.

Proof.

Part (1). By the assumption,[4] U is $[A]$-acyclic and $f : dom(Q) \to dom(U)$ is a homomorphism such that $f(Q) \subseteq U$. Since U is $[A]$-acyclic, it has a (possibly infinite) $[A]$-join forest $T = \langle\langle V, E\rangle, \lambda\rangle$. We assume, without loss of generality, that distinct vertices u, v of T have different labels, i.e., $\lambda(u) \neq \lambda(v)$. This assumption can be made by removing all subforests rooted at nodes labeled by duplicate atoms. Let T_Q be the finite subforest of T

3. The join graph of H has the atoms as nodes. An edge between two atoms exists iff the atoms share at least one variable.

4. One may be tempted to conjecture that $W = f(Q)$, but this does not work because acyclicity (and thus also $[A]$-acyclicity) is not a hereditary property: it may well be the case that U is acyclic, while the subset $f(Q) \subseteq U$ is not. However, taking $W = f(Q)$ works in case of arities at most 2.

that contains all ancestors in T of nodes s such that $\lambda(s) \in f(Q)$. Let $F = \langle\langle V', E'\rangle, \lambda'\rangle$ be the forest obtained from T as follows.

- $V' = \{v \in V \mid \lambda(v) \in f(Q)\} \cup K$, where K is the set of all vertices of T_Q that have at least two children.
- If $v, w \in V'$ then there is an edge from v to w in E' iff w is a descendant of v in T, and if the unique shortest path from v to w in T does not contain any other node from V'.
- Finally, for each $v \in V'$, $\lambda'(v) = \lambda(v)$.

Let us define $W = \lambda(V')$. We claim that the forest F is an $[A]$-join forest of W. Since Condition *(1)* of Definition 3.8 ($[S]$-join forest) is immediately satisfied, it suffices to show Condition *(2)*, that is, that F satisfies the $[A]$-connectedness condition. Assume, for any pair of distinct vertices v_1 and v_2 in F, that for some value $b \in dom(U) - A$ it holds $b \in dom(\lambda'(v_1)) \cap dom(\lambda'(v_2))$. In order to prove the aforementioned $[A]$-connectedness condition, we need to show that there is at least one path in F between v_1 and v_2 (here we view F as a undirected graph), and that for every node $v \in V'$ lying on each such path we have $b \in dom(\lambda'(v))$. By construction of F, v_1 and v_2 are connected in T and v lies on the (unique) path between v_1 and v_2 in T. Since T is an $[A]$-join forest, we have $b \in dom(\lambda(v)) = dom(\lambda'(v))$. Thus F is an $[A]$-join forest of W.

Moreover, by construction of F, the number of children of each inner vertex of F is at least 2, and F has at most $|Q|$ leaves. It follows that F has at most $2|Q| - 1$ vertices. Therefore W is an $[A]$-acyclic set of atoms such that $|W| \leqslant 2|Q|$ and $W \supseteq f(Q)$.

Part (2). Q can be extended to Q^+ as follows. For each atom $r(t_1, \ldots, t_k)$ in $W - f(Q)$, add to Q a new query atom $r(\xi_1, \ldots, \xi_k)$ such that for each $1 \leqslant i \leqslant k$, ξ_i is a newly invented variable. Obviously, $W \models Q^+$ and thus there is a homomorphism g extending f such that $g(Q^+) = W$. Moreover, by construction $|Q^+| < 2|Q|$. □

Lemma 5.7. *Let G be an $[A]$-acyclic instance and let G' be an instance obtained from G by eliminating a set S of atoms where $dom(S) \subseteq A$. Then G' is $[A]$-acyclic.*

Proof. If $T = \langle\langle V, E\rangle, \lambda\rangle$ is an $[A]$-join forest for G then an $[A]$-join forest T' for G' can be obtained from G by repeatedly eliminating each vertex v from T where $\lambda(v) \in S$. By construction, each atom $\underline{e}$ eliminated from G in this way has the property that $dom(\underline{e}) \subseteq A$. Hence, for every value $b \in dom(G) - A$, the node $u \in V$ such that $\lambda(u) = \underline{e}$ cannot belong to the induced (connected) subtree $\{v \in V \mid b \in dom(\lambda(v))\}$. We thus get that G' enjoys the $[A]$-connectedness property. □

The following Lemma will be used as a main tool in the subsequent complexity analysis.

Lemma 5.8 (Squid Lemma)**.** *Let Σ be a weakly guarded set of TGDs on a schema $\mathcal{R}$, D a database for $\mathcal{R}$, and Q a Boolean conjunctive query. Then $chase(D, \Sigma) \models Q$ iff there is a squid decomposition $\delta = (Q^+, h, H, T)$ and a homomorphism $\theta : dom(h(Q^+)) \to dom(chase(D, \Sigma))$ such that: (i) $\theta(H) \subseteq chase^{\perp}(D, \Sigma)$, and (ii) $\theta(T) \subseteq chase^{+}(D, \Sigma)$.*

Proof. *"If"*. If there is a squid decomposition $\delta = (Q^+, h, H, T)$ of Q and a homomorphism θ as described, then the composition $\theta \circ h$ is a homomorphism such that $(\theta \circ h)(Q^+) =$

$\theta(h(Q^+)) \subseteq \mathit{chase}(D,\Sigma)$. Hence, $\mathit{chase}(D,\Sigma) \models Q^+$ and, by Lemma 5.3, $\mathit{chase}(D,\Sigma) \models Q$.

"Only if". Assume $U = \mathit{chase}(D,\Sigma) \models Q$. Then, there exists a homomorphism $f : \mathit{vars}(Q) \to \mathit{dom}(U)$ with $f(Q) \subseteq \mathit{chase}(D,\Sigma)$. By Lemma 3.11, $\mathit{chase}^+(D,\Sigma)$ is $[\mathit{dom}(D)]$-acyclic. By Lemma 5.6, it then follows that there is a Boolean query Q^+ with $< 2|Q|$ atoms, such that all atoms of Q are contained in Q^+, and there is a homomorphism $g : \mathit{dom}(Q^+) \to \mathit{dom}(U)$ with $g(Q^+) \subseteq U$, such that $g(Q^+)$ is $[\mathit{dom}(D)]$-acyclic.

Partition $\mathit{vars}(Q^+)$ into two sets $\mathit{vars}^\perp(Q^+)$ and $\mathit{vars}^+(Q^+)$ as follows:

- $\mathit{vars}^\perp(Q^+) = \{X \in \mathit{vars}(Q^+) \mid g(X) \in \mathit{dom}(D)\}$
- $\mathit{vars}^+(Q^+) = \mathit{vars}(Q^+) - \mathit{vars}^\perp(Q^+)$.

Define a mapping $h : \mathit{vars}(Q^+) \to \mathit{vars}(Q^+)$ as follows. For each $X \in \mathit{vars}(Q^+)$, let $h(X)$ be the lexicographically first variable in the set $\{Y \in \mathit{vars}(Q^+) \mid g(Y) = g(X)\}$. Let us define V_δ as $V_\delta = h(\mathit{vars}^\perp(Q^+))$. Moreover, let H be the set of all those atoms $\underline{a}$ of $h(Q^+)$, such that $\mathit{vars}(\underline{a}) \subseteq V_\delta = h(\mathit{vars}^\perp(Q^+))$, and let $T = h(Q^+) - H$. Note that, by definition of H, $g(H) \subseteq \mathit{chase}^\perp(D,\Sigma)$ and, by definition of T, $g(T) \subseteq \mathit{chase}^+(D,\Sigma)$. Let θ be the restriction of g to $\mathit{dom}(h(Q^+))$. Clearly, θ, h, H, and T fulfill the conditions *(i)* and *(ii)* of the statement of this lemma. It thus remains to prove that $\delta = (Q^+, h, H, T)$ is actually a squid decomposition of Q. For this, we only need to show that T is $[V_\delta]$-acyclic. To prove this, first observe that for each pair of variables X, Y in $\mathit{vars}(Q^+)$ such that $g(X) = g(Y)$ we have $h(X) = h(Y)$. Therefore θ is, by construction, a bijection between $h(\mathit{dom}(Q^+))$ and $\mathit{dom}(\theta(Q^+))$. In particular, $T \subseteq h(Q^+)$ is isomorphic to $\theta(T)$ via the restriction θ_T of θ to $\mathit{dom}(T)$. Since $\theta_T(T) = \theta(T)$ is obtained from the $[\mathit{dom}(D)]$-acyclic instance $\theta(Q^+)$ by eliminating only atoms all of whose arguments are in $\mathit{dom}(D)$ (namely the atoms in $\theta(H)$), by Lemma 5.7, $\theta_T(T)$ is itself $[\mathit{dom}(D)]$-acyclic and, therefore, trivially also $[\mathit{dom}(D) \cap \mathit{dom}(\theta(T))]$-acyclic. Now, since for every $X \in \mathit{dom}(T)$ it holds that $X \in V_\delta$ iff $\theta_T(X) \in \mathit{dom}(D)$, it immediately follows that, since $\theta_T(T)$ is $[\mathit{dom}(D)]$-acyclic, T is $[V_\delta]$-acyclic. $\square$

5.2 Clouds and the Complexity of Query Answering under WGTGDs

The goal of this subsection is to prove the following theorem:

Theorem 5.9. *Let Σ be a weakly guarded set of TGDs, D a database for a schema $\mathcal{R}$, and Q a Boolean conjunctive query. The problem of determining whether $D \cup \Sigma \models Q$ or, equivalently, whether $\mathit{chase}(D,\Sigma) \models Q$ is in* EXPTIME *in case of bounded arity, and in* 2EXPTIME *in general.*

For the general case (of unbounded arities), we first outline a short high-level proof aimed at specialists in Computational Logic. This proof makes sophisticated use of previous results. We will then give a much longer, self-contained proof, that works for both the general case and the case of bounded arities. The self-contained proof also introduces some concepts that will be used in the following sections.

High Level Proof Sketch of Theorem 5.9 (General Case). We transform the original problem instance (D, Σ, Q) into a guarded first-order theory $\Gamma = \tau(D, \Sigma, Q)$ such that $\mathit{chase}(D,\Sigma) \models$

Q iff Γ is unsatisfiable. The signature σ of Γ uses Σ as the set of constants plus a constant for each element of $dom(D)$. Moreover, σ includes all predicate symbols occurring in D, Σ, or Q, plus a special nullary (i.e., propositional) predicate symbol q.

Γ contains all ground facts of D, plus all instances of each rule $r \in \Sigma$ obtained from r by replacing all variables of r that occur in non-affected positions with constants. Note that the resulting rules are universally quantified guarded sentences. Moreover, for each squid decomposition $\delta = (Q^+, h, H, T, V_\delta)$, and each possible replacement θ of the set of variables V_δ by constants of the signature σ, Γ contains a guarded sentence ϕ_δ^θ obtained as follows. Notice that $Q_\delta^\theta := \theta(H) \cup T$ is a Boolean acyclic conjunctive query. By the results of Gottlob, Leone, and Scarcello (2003),[5] Q_δ^θ can thus be rewritten (in polynomial time) into an equivalent guarded sentence ψ_δ^θ. We define ϕ_δ^θ to be $(\psi_\delta^\theta \rightarrow q)$, which is obviously guarded, too. Let Γ^- denote the sentences of Γ mentioned so far. From this construction and the Squid Lemma (Lemma 5.8), it follows that $chase(D, \Sigma) \models Q$ iff $\Gamma^- \models q$. Now let $\Gamma = \Gamma^- \cup \{\neg q\}$. Obviously, Γ is unsatisfiable iff $chase(D, \Sigma) \models Q$.

Note that the reduction τ is an arity-preserving EXPTIME-reduction. Let t be an exponential upper bound on the runtime required by reduction τ (and thus also on the size of $\tau(D, \Sigma, Q)$). A deterministic version of the algorithm in the work by Grädel (1999) for deciding whether a guarded theory of unbounded arity is satisfiable or unsatisfiable runs in double-exponential time $O(2^{O(s \cdot w^w)})$, where s is the size of the theory and w is its maximum predicate arity. Therefore, the overall runtime of first computing $\Gamma = \tau(D, \Sigma, Q)$ for an input (D, Σ, Q) of size n and maximum arity w, and then checking whether Γ is unsatisfiable is $O(2^{O(t(n) \cdot w^w)})$, which is still only double-exponential. Deciding $D \cup \Sigma \models Q$ is thus in 2EXPTIME. □

Note that in case of bounded w, a similar proof does not provide an EXPTIME bound, as $2^{t(n) \cdot w^w}$ would still be doubly exponential due to the exponential term $t(n)$, even if w is constant. Actually, as noted by Bárány, Gottlob, and Otto (2010), evaluating non-atomic conjunctive queries against guarded first-order theories of bounded predicate arity is in fact 2EXPTIME-complete. Surprisingly, this remains true even for guarded theories $D \cup \Sigma$ where D is a (variable) database and Σ a *fixed* guarded theory of a very simple form involving disjunctions (Bourhis, Morak, & Pieris, 2013). We therefore needed to develop different proof ideas.

In the rest of this section we present an independent and self-contained proof of Theorem 5.9 by developing tools for analyzing the complexity of query answering under WGTGDs. To this end we introduce the notion of a *cloud* of an atom $\underline{a}$ in the chase of a database D under a set Σ of WGTGDs. Intuitively, the cloud of $\underline{a}$ is the set of atoms of $chase(D, \Sigma)$ whose arguments belong to $dom(\underline{a}) \cup dom(D)$. In other words, the atoms in the cloud cannot have nulls that do not appear in $\underline{a}$. The cloud is important because we will show that the subtree of $\mathrm{gcf}(D, \Sigma)$ rooted in $\underline{a}$ depends only on $\underline{a}$ and its cloud.

Definition 5.10. Let Σ be a weakly guarded set of TGDs on a schema $\mathcal{R}$ and D be a database for $\mathcal{R}$. For every atom $\underline{a} \in chase(D, \Sigma)$ the *cloud* of $\underline{a}$ with respect to Σ and D is the following set: $cloud(D, \Sigma, \underline{a}) = \{\underline{b} \in chase(D, \Sigma) \mid dom(\underline{b}) \subseteq dom(\underline{a}) \cup dom(D)\}$. Notice

5. See Theorem 3 in that paper, its proof, the remark after that proof, and Corollary 3.

that for every atom $\underline{a} \in \mathit{chase}(D, \Sigma)$ we have $D \subseteq \mathit{cloud}(D, \Sigma, \underline{a})$. Moreover, we define

$$\begin{array}{rcl} \mathit{clouds}(D, \Sigma) & = & \{\mathit{cloud}(D, \Sigma, \underline{a}) \mid \underline{a} \in \mathit{chase}(D, \Sigma)\} \\ \mathit{clouds}^{+}(D, \Sigma) & = & \{(\underline{a}, \mathit{cloud}(D, \Sigma, \underline{a})) \mid \underline{a} \in \mathit{chase}(D, \Sigma)\} \end{array}$$

Any subset $S \subseteq \mathit{cloud}(D, \Sigma, \underline{a})$ is called a *subcloud* of $\underline{a}$ (with respect to Σ and D). The set of all subclouds of an atom $\underline{a}$ is denoted by $\mathit{subclouds}(D, \Sigma, \underline{a})$. Finally, we define $\mathit{subclouds}^{+}(D, \Sigma) = \{(\underline{a}, C) \mid \underline{a} \in \mathit{chase}(D, \Sigma) \text{ and } C \subseteq \mathit{cloud}(D, \Sigma, \underline{a})\}$.

Definition 5.11. Let B be an instance (possibly with nulls) over a schema, $\mathcal{R}$, and D be a database over $\mathcal{R}$. Let α and β be atoms from the Herbrand Base $\mathit{HB}(B)$. We say that α and β are *D-isomorphic*, denoted $\alpha \simeq_D \beta$, or simply $\alpha \simeq \beta$ in case D is understood, if there is a bijective homomorphism[6] $f : \mathit{dom}(\alpha) \to \mathit{dom}(\beta)$ such that $f(\alpha) = \beta$ (and thus also $f^{-1}(\beta) = \alpha$). This definition extends directly to the cases when α and β are sets of atoms or atom-set pairs (in a similar fashion as in $\mathit{clouds}^{+}(D, \Sigma)$).

Example 5.12. If $\{a, b\} \subseteq \mathit{dom}(D)$ and $\{\zeta_1, \zeta_2, \zeta_3, \zeta_4\} \subseteq \Delta_N$, we have: $p(a, \zeta_1, \zeta_2) \simeq p(a, \zeta_3, \zeta_4)$ and $(p(a, \zeta_3), \{q(a, \zeta_3), q(\zeta_3, \zeta_3), r(\zeta_3)\}) \simeq (p(a, \zeta_1), \{q(a, \zeta_1), q(\zeta_1, \zeta_1), r(\zeta_1)\})$. On the other hand, $p(a, \zeta_1, \zeta_2) \not\simeq p(a, \zeta_1, \zeta_1)$ and $p(a, \zeta_1, \zeta_2) \not\simeq p(\zeta_3, \zeta_1, \zeta_2)$. □

Lemma 5.13. *Given a database D for a schema $\mathcal{R}$ and an instance B for $\mathcal{R}$, the D-isomorphism relation $\simeq_D$ on $\mathit{HB}(B)$ (resp., $2^{\mathit{HB}(B)}$ or $\mathit{HB}(B) \times 2^{\mathit{HB}(B)}$, as in Definition 5.11) is an equivalence relation.*

The above lemma follows directly from the definitions; it lets us define, for every set A of atoms of $\mathit{HB}(B)$, the quotiont set $A/_{\simeq_D}$ with respect to the above defined equivalence relation $\simeq_D$. Such notion of quotient set naturally extends to sets of sets of atoms such as $\mathit{clouds}(D, \Sigma)$, or sequences (pairs, in particular) thereof.

Lemma 5.14. *Let Σ be a weakly guarded set of TGDs and let D be a database for a schema $\mathcal{R}$. Let $|\mathcal{R}|$ denote the number of predicate symbols in $\mathcal{R}$, and w be the maximum arity of the symbols in $\mathcal{R}$. Then:*

(1) For every atom $\underline{a} \in \mathit{chase}(D, \Sigma)$, we have $|\mathit{cloud}(D, \Sigma, \underline{a})| \leqslant |\mathcal{R}| \cdot (|\mathit{dom}(D)| + w)^w$. Thus, $\mathit{cloud}(D, \Sigma, \underline{a})$ is polynomial in the size of the database D if the arity w is bounded and exponential otherwise.
(2) For each atom $\underline{a} \in \mathit{chase}(D, \Sigma)$, $|\mathit{subclouds}(D, \Sigma, \underline{a})| \leqslant 2^{|\mathcal{R}| \cdot (|\mathit{dom}(D)| + w)^w}$.
(3) $|\mathit{clouds}(D, \Sigma)/_{\simeq}| \leqslant 2^{|\mathcal{R}| \cdot (|\mathit{dom}(D)| + w)^w}$, i.e., there are—up to isomorphism—at most exponentially many possible clouds or subclouds in a chase, if the arity w is bounded, otherwise doubly exponentially many.
(4) $|\mathit{clouds}^{+}(D, \Sigma)/_{\simeq}| \leqslant |\mathit{subclouds}^{+}(D, \Sigma)/_{\simeq}| \leqslant |\mathcal{R}| \cdot (|\mathit{dom}(D)| + w)^w \cdot 2^{|\mathcal{R}| \cdot (|\mathit{dom}(D)| + w)^w}$.

Proof. The claims are proved by combinatorial arguments as follows.

(1) All distinct atoms in a cloud are obtained by placing the symbols of $\underline{a}$, plus possibly symbols from $\mathit{dom}(D)$, in at most w arguments of some predicate symbol in $\mathcal{R}$. For each predicate in $\mathcal{R}$, the number of symbols to be thus placed is $|\mathit{dom}(D)| + w$.

6. Recall that, by definition, the restriction of a homomorphism to $\mathit{dom}(D)$ is the identity mapping.

(2) The different ways we can choose $subclouds(D, \Sigma, \underline{a})$ clearly determines the set of all subsets of $cloud(D, \Sigma, \underline{a})$.

(3) It is easy to see that the size of the set of all non-pairwise-isomorphic clouds in the chase is bounded by the number of possible subclouds of a fixed atom.

(4) Here, we are counting the number of all possible subclouds, each associated with its "generating" atom. The inequality holds because, once we choose all non-pairwise-isomorphic clouds, each of their possible generating atoms can have as arguments only $|dom(D)| + w$ symbols with which to construct the subclouds. □

Definition 5.15. Given a database D and a set of WGTGDs, let $\underline{a}$ be an atom in $chase(D, \Sigma)$. We define the following notions:

- $\underline{a}_{\downarrow}$ is the set of all atoms that label nodes of the subtrees of $\text{gcf}(D, \Sigma)$ rooted in $\underline{a}$;
- $\nabla\underline{a} = \underline{a}_{\downarrow} \cup cloud(D, \Sigma, \underline{a})$;
- if S is a subset of atoms in $\text{gcf}(D, \Sigma)$, then $\text{gcf}[\underline{a}, S]$[7] is inductively defined as follows:
  (i) $S \cup \{\underline{a}\} \subseteq \text{gcf}[\underline{a}, S]$;
  (ii) $\underline{b} \in \text{gcf}[\underline{a}, S]$ if $\underline{b} \in \underline{a}_{\downarrow}$, and $\underline{b}$ is obtained via the chase rule applied using a TGD with body Φ and head-atom $\underline{\psi}$, and a homomorphism θ, such that $\theta(\underline{\psi}) = \underline{b}$ and $\theta(\Phi) \subseteq \text{gcf}[\underline{a}, S]$.

Theorem 5.16. *If D is a database for a schema $\mathcal{R}$, Σ is a weakly guarded set of TGDs, and $\underline{a} \in chase(D, \Sigma)$, then* $\nabla\underline{a} = \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$.

Proof. By the definitions of $\nabla\underline{a}$ and $\text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$, we have $\text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})] \subseteq \nabla\underline{a}$. It remains to show the converse inclusion: $\nabla\underline{a} \subseteq \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$. Define $\text{level}_{\underline{a}}(\underline{a}) = 0$ and for each fact $\underline{b} \in cloud(D, \Sigma, \underline{a}) - \nabla\underline{a}$ we also define $\text{level}_{\underline{a}}(\underline{b}) = 0$. For every other atom $\underline{c} \in \underline{a}_{\downarrow}$, $\text{level}_{\underline{a}}(\underline{c})$ is defined as the distance (i.e., the length of the path) from $\underline{a}$ to $\underline{c}$ in $\text{gcf}(D, \Sigma)$.

We first show the following facts in parallel by induction on $\text{level}_{\underline{a}}(\underline{b})$:

(1) If $\underline{b} \in \nabla\underline{a}$ then $cloud(D, \Sigma, \underline{b}) \subseteq \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$.

(2) If $\underline{b} \in \nabla\underline{a}$ then $\underline{b} \in \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$.

Statement *(2)* above is the converse inclusion we are after.

Induction basis. If $\text{level}_{\underline{a}}(\underline{b}) = 0$, we have either *(a)* $\underline{b} \in cloud(D, \Sigma, \underline{a}) - \{\underline{a}\}$, or *(b)* $\underline{b} = \underline{a}$. In case *(a)*, $cloud(D, \Sigma, \underline{a}) \subseteq \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$ and therefore $\underline{b} \in \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$, which proves *(1)*. Moreover, since $\underline{b} \in cloud(D, \Sigma, \underline{a})$, $\underline{b}$ cannot contain more labeled nulls than $\underline{a}$, so $dom(\underline{b}) - dom(D) \subseteq dom(\underline{a}) - dom(D)$. Therefore $cloud(D, \Sigma, \underline{b}) \subseteq cloud(D, \Sigma, \underline{a}) \subseteq \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$, which proves *(2)*. In case *(b)*, $\underline{b} = \underline{a}$ and thus $cloud(D, \Sigma, \underline{a}) = cloud(D, \Sigma, \underline{b}) \subseteq \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$, which proves *(1)*. Since $\underline{b} = \underline{a} \in \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$, *(2)* follows as well.

Induction step. Assume that *(1)* and *(2)* are satisfied for all $\underline{c} \in \nabla\underline{a}$ such that $\text{level}_{\underline{a}}(\underline{c}) \leqslant i$ and assume $\text{level}_{\underline{a}}(\underline{b}) = i + 1$, where $i \geqslant 0$. The atom $\underline{b}$ is produced by a TGD whose guard $\underline{g}$ matches some atom $\underline{b}^{-}$ at level i, which is, by the induction hypothesis, in $\text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$. The body atoms of such a TGD then match atoms whose arguments

7. D and Σ are implicit here, to avoid clutter.

must be in $cloud(D, \Sigma, \underline{b})$ and thus also in $\text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$, again by the induction hypothesis. Therefore, *(2)* holds for $\underline{b}$. To show *(1)*, consider an atom $\underline{b}' \in cloud(D, \Sigma, \underline{b})$. In case $dom(\underline{b}') \subseteq dom(\underline{b}^-)$, we have $cloud(D, \Sigma, \underline{b}') \subseteq cloud(D, \Sigma, \underline{b}^-) \subseteq \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$. Otherwise, $\underline{b}'$ contains at least one new labeled null that was introduced during the generation of $\underline{b}$. Given that Σ is a weakly guarded set and each labeled null in Δ_N is introduced only once in the chase, there must be a path from $\underline{b}$ to $\underline{b}'$ in $\text{gcf}(D, \Sigma)$ (and therefore also in $\nabla\underline{b}$). A simple additional induction on $\text{level}_{\underline{b}}(\underline{b}')$ shows that all the applications of TGDs on that path must have been fired on elements of $\text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$ only. Therefore, $\underline{b}' \in \text{gcf}[\underline{a}, cloud(D, \Sigma, \underline{a})]$, which proves *(1)*. $\square$

The corollary below follows directly from the above theorem.

Corollary 5.17. *If D is a database for a schema $\mathcal{R}$, Σ is a weakly guarded set of TGDs, $\underline{a}, \underline{b} \in chase(D, \Sigma)$, and $(\underline{a}, cloud(D, \Sigma, \underline{a})) \simeq (\underline{b}, cloud(D, \Sigma, \underline{b}))$, then $\nabla\underline{a} \simeq \nabla\underline{b}$.*

Definition 5.18. Let D be a database and $\underline{a}$ an atom. The *canonical renaming* $can_{\underline{a}} : dom(\underline{a}) \cup dom(D) \to \Delta_{\underline{a}} \cup dom(D)$, where $\Delta_{\underline{a}} = \{\xi_1, \ldots, \xi_h\} \subset \Delta_N$ is a set of labeled nulls *not* appearing in $\underline{a}$, is a 1-1 substitution that maps each element of $dom(D)$ into itself and each null-argument of $\underline{a}$ to the first unused element $\xi_i \in \Delta_{\underline{a}}$. If $S \subseteq cloud(D, \Sigma, \underline{a})$ then $can_{\underline{a}}(S)$ is well-defined and the pair $(can_{\underline{a}}(\underline{a}), can_{\underline{a}}(S))$ will be denoted by $can(\underline{a}, S)$.

Example 5.19. Let $\underline{a} = g(d, \zeta_1, \zeta_2, \zeta_1)$ and $S = \{p(\zeta_1), r(\zeta_2, \zeta_2), s(\zeta_1, \zeta_2, b)\}$, where $\{d, b\} \subseteq dom(D)$ and $\{\zeta_1, \zeta_2\} \subseteq \Delta_N$. Then $can_{\underline{a}}(\underline{a}) = g(d, \xi_1, \xi_2, \xi_1)$, and $can_{\underline{a}}(S) = \{p(\xi_1), r(\xi_2, \xi_2), s(\xi_1, \xi_2, b)\}$. $\square$

Definition 5.20. If D is a database for a schema $\mathcal{R}$, Σ is a weakly guarded set of TGDs on $\mathcal{R}$, S is a set of atoms and $\underline{a} \in S$, then we write $(D, \Sigma, \underline{a}, S) \models Q$ iff there exists a homomorphism θ such that $\theta(Q) \subseteq S \cup \underline{a}_{\downarrow}$.

The following result straightforwardly follows from Theorem 5.16 and the previous definitions.

Corollary 5.21. *If D is a database for a schema $\mathcal{R}$, Σ is a weakly guarded set of TGDs, $\underline{a} \in chase(D, \Sigma)$, and Q is a Boolean conjunctive query, then the following statements are equivalent:*

(1) $\nabla\underline{a} \models Q$
(2) $(D, \Sigma, \underline{a}, cloud(D, \Sigma, \underline{a})) \models Q$
(3) $(D, \Sigma, can_{\underline{a}}(a), can_{\underline{a}}(cloud(D, \Sigma, \underline{a}))) \models Q$
(4) there is a subset $S' \subseteq cloud(D, \Sigma, \underline{a})$ such that $(D, \Sigma, can_{\underline{a}}(\underline{a}), can_{\underline{a}}(S')) \models Q$.

We will use the pair $can(\underline{a}, cloud(D, \Sigma, \underline{a}))$ as a unique canonical representative of the equivalence class $\{(\underline{b}, cloud(D, \Sigma, \underline{b})) \mid (\underline{b}, cloud(D, \Sigma, \underline{b})) \simeq (\underline{a}, cloud(D, \Sigma, \underline{a}))\}$ in $clouds^+(D, \Sigma)$. Therefore, the set $\{can(\underline{a}, cloud(D, \Sigma, \underline{a})) \mid \underline{a} \in chase(D, \Sigma)\}$ and the quotient set $clouds^+(D, \Sigma)/_{\simeq}$ are isomorphic. Note that, by Lemma 5.14, these sets are finite and have size exponential in $|D| + |\Sigma|$ if the schema is fixed (and double exponential otherwise).

Now, given a database D for a schema $\mathcal{R}$, a weakly guarded set of TGDs Σ on $\mathcal{R}$, and an *atomic* Boolean conjunctive query Q, we describe an alternating algorithm $\mathsf{Acheck}(D, \Sigma, Q)$

that decides whether $D \cup \Sigma \models Q$. We assume that Q has the form $\exists Y_1, \ldots, Y_\ell, q(t_1, t_2, \ldots, t_r)$, where the $t_1, \ldots, t_r$, with $r \geqslant \ell$, are terms (constants or variables) in $dom(D) \cup \{Y_1, Y_2, \ldots, Y_\ell\}$.

The algorithm Acheck returns "true" if it accepts some configuration, according to the criteria explained below; otherwise, it returns "*false*". Acheck uses tuples of the form $(\underline{a}, S, S', \prec, \underline{b})$ as its basic data structures (*configurations*). Intuitively, each such configuration corresponds to an atom $\underline{a}$ derived at some step of the chase computation together with a set S' of already derived atoms belonging the cloud of $\underline{a}$. The informal meaning of the parameters of a configuration is as follows.

(1) $\underline{a}$ is the root atom of the chase subtree under consideration.

(2) $S \subseteq cloud(D, \Sigma, \underline{a})$; S is intuitively a subcloud containing a set of atoms of $cloud(D, \Sigma, \underline{a})$ that, while computing $chase(D, \Sigma)$, are originally derived *outside* the subtree of the guarded chase forest rooted in $\underline{a}$ (and are thus outside the subtree rooted in $\underline{a}$ of $\mathrm{rgcf}(D, \Sigma)$). We expect these atoms to serve as "side atoms" (i.e., atoms matching non-guard atoms of a TGD) when deriving the desired atom $\underline{b}$ starting at $\underline{a}$.

(3) S' contains, at every step in the computation, the subset of $cloud(D, \Sigma, \underline{a})$ that has been computed so far, or can be assumed to be valid, as it will be verified in another branch of the computation.

(4) $\prec$ is a total ordering of the atoms in S consistent with the order in which the atoms of S are proved by the algorithm (by simulating the chase procedure).

(5) $\underline{b}$ is an atom that needs to be derived. In some cases (namely, on the "main" path in the proof tree developed by Acheck), the algorithm will not try to derive a specific atom, but will just match the query atom $q(t_1, \ldots, t_r)$ against the atoms of that path. In that case, we use the symbol $\star$ in place of $\underline{b}$.

We are now ready to describe the algorithm Acheck at a sufficiently detailed level. However, we omit many low-level details.

Acheck first checks if $D \models Q$. If so, Acheck returns "true" and halts. Otherwise, the algorithm attempts to guess a path, the so called *main branch*, that contains an atom $\underline{q}$ that is an instance of Q. This is done as follows.

Initialization. The algorithm Acheck starts at D and guesses some atom $\underline{a} \in D$, which it will expand into a main branch that will eventually lead to an atom $\underline{q}$ matching the query Q. To this end, the algorithm guesses a set $S \subseteq cloud(D, \Sigma, \underline{a})$ and a total order $\prec$ on S, and then generates the configuration $c_0 = (\underline{a}, S, S', \prec, \star)$. The set S' is initialized as $S' = S$.

Form of a configuration—additional specifications. In each configuration, the set set S is implicitly partitioned into two sets $S^\perp$ and S^+, where $S^\perp \subseteq D$ and $S^+ = \{\underline{a}_1, \underline{a}_2, \ldots, \underline{a}_k\}$ is disjoint from D. The total order $\prec$ is such that all elements of $S^\perp$ precede those of S^+. On S^+, $\prec$ is defined as $\underline{a}_1 \prec \underline{a}_2 \prec \cdots \prec \underline{a} \prec \cdots \prec \underline{a}_k$.

Summary of tasks Acheck performs for each configuration. Assume the Acheck algorithm generates a configuration $c = (\underline{a}, S, S', \prec, \underline{b})$, where $\underline{b}$ might be $\star$. Acheck then performs the following tasks on c:

- Acheck verifies that the guessed set S of c is actually a subset of $cloud(D, \Sigma, \underline{a})$. This is achieved by a massive universal branching that will be described below under the heading “Universal Branching”. Let us, however, anticipate here how it works, as this may contribute to the understanding of the other steps. Acheck will verify that each of the atoms $\underline{a}_1, \ldots, \underline{a}_k$ is in $chase(D, \Sigma)$, where, for each $i \in \{1, \ldots, k\}$, the proof of $\underline{a}_i \in chase(D, \Sigma)$ can use as premises only the atoms of S that precede $\underline{a}_i$, according to $\prec$. The algorithm thus finds suitable atoms $\underline{d}_1, \ldots, \underline{d}_k \in D$ and builds proof trees for $\underline{a}_1, \ldots, \underline{a}_k$. For each $1 \leqslant i \leqslant k$, it generates configurations of the form $(\underline{d}_i, S, S^{\perp} \cup \{\underline{a}_1, \underline{a}_2, \ldots, \underline{a}_{i-1}\}, \prec, \underline{a}_i)$. Each such configuration will be used as a starting point in a proof of $\underline{a}_i \in chase(D, \Sigma)$ assuming that $\underline{a}_1, \ldots, \underline{a}_{i-1} \in chase(D, \Sigma)$ has already been established. Acheck thus simulates a sequential proof of all atoms of $cloud(D, \Sigma, \underline{a})$ that are in S via a parallel universal branching from c.

- Acheck tests whether c is a final configuration (i.e., an accepting or rejecting one). This is described under the heading “Test for final Configuration” below.
- If c is not a final configuration of Acheck, this means that its first component $\underline{a}$ is not yet the one that will be matched to $\underline{b}$ (or the query, if $\underline{b} = \star$). Acheck then “moves down” the chase tree by one step by replacing $\underline{a}$ with a child of $\underline{a}$. This step is described under the heading “Existential Branching”.

In the following, let $c = (\underline{a}, S, S', \prec, \underline{b})$ be a configuration, where $\underline{b}$ may be $\star$.

Test for final configuration. If $\underline{b} \in D$, then Acheck accepts this configuration, and does not expand it further. If $\underline{b} = \star$, then Acheck checks (via a simple subroutine) whether Q matches $\underline{a}$, i.e., if $\underline{a}$ is a homomorphic image of the query atom $q(t_1, \ldots, t_r)$. If so, Acheck accepts c (and thus returns “true”) and does not expand it further. If $\underline{b} \neq \star$, Acheck checks whether $\underline{a} = \underline{b}$. If this is true, then Acheck accepts the configuration c and does not expand it further. Otherwise, the configuration tree is expanded as described next.

Existential Branching. Acheck guesses a TGD $\rho \in \Sigma$ having body Φ and head-atom $\underline{\psi}$, and whose guard $\underline{g}$ matches $\underline{a}$ via some substitution θ (that is, $\theta(\underline{g}) = \underline{a}$) such that $\theta(\Phi) \subseteq S'$. $\theta(\underline{\psi})$ then corresponds to a newly generated atom (possibly containing some fresh labeled nulls in Δ_N). Note that, if no such guess can be made, this existential branching automatically fails and Acheck returns *false*. To define the configuration c_1 that Acheck creates out of c, we first introduce an intermediate auxiliary configuration $\hat{c} = (\hat{\underline{a}}, \hat{S}, \hat{S}', \hat{\prec}, \hat{\underline{b}})$, where:

(a) $\hat{\underline{a}} = \theta(\underline{\psi})$ is the new atom generated by the application of ρ with the substitution θ.

(b) $\hat{S}$ contains $\hat{\underline{a}}$ and each atom $\underline{d}$ of S such that $dom(\underline{d}) \subseteq dom(\hat{\underline{a}}) \cup dom(D)$. Thus, in addition to the new atom $\hat{\underline{a}}$, $\hat{S}$ *inherits* all atoms that were in the subcloud S of the parent configuration c that are “compatible” with $\hat{\underline{a}}$. In addition, $\hat{S}$ includes a set $\mathsf{newatoms}(\hat{c})$ of new atoms that are guessed by the Acheck algorithm. All arguments of each atom of $\mathsf{newatoms}(\hat{c})$ must be elements of the set $dom(\hat{\underline{a}}) \cup dom(D)$.

(c) $\hat{S}' = \hat{S}$.

(d) $\hat{\prec}$ is a total order on $\hat{S}'$ obtained from $\prec$ by eliminating all atoms in $S - \hat{S}$ and by ordering the atoms from $\mathsf{newatoms}(\hat{c})$ after all the atoms from the set $\mathsf{oldproven}(\hat{c}) = \hat{S}' \cap S'$ (these are assumed to have already been proven at the parent configuration c).

(e) $\hat{\underline{b}}$ is defined as $\hat{\underline{b}} = \underline{b}$.

Next, Acheck constructs the configuration c_1 out of $\hat{c}$ by canonicalization: $c_1 = can_{\hat{\underline{a}}}(\hat{c})$, that is $c_1 = (can_{\hat{\underline{a}}}(\hat{\underline{a}}),\ can_{\hat{\underline{a}}}(\hat{S}),\ can_{\hat{\underline{a}}}(\hat{S}'),\ can_{\hat{\underline{a}}}(\hat{\prec}),\ can_{\hat{\underline{a}}}(\hat{\underline{b}}))$, where $can_{\hat{\underline{a}}}(\hat{\prec})$ is the total order on the atoms in $can_{\hat{\underline{a}}}(\hat{S}')$ derived from $\hat{\prec}$.

Intuitively, c_1 is the "main" child of c on the way to deriving the query atom $q(t_1, \ldots, t_r)$ assuming that all atoms of the guessed subcloud S are derivable.

Universal Branching. In the above generated configuration $\hat{c}$, the set $\hat{S}'$ is equal to $\hat{S}$. As already said, this means that it is assumed for that configuration that the set of atoms $\hat{S}$ is derivable. To verify that this is indeed the case, Acheck generates in parallel, using universal computation branching, a set of auxiliary configurations for proving that all the guessed atoms in $can_{\underline{\hat{a}}}(\mathsf{newatoms}(\hat{c}))$ are indeed derivable through the chase of D with respect to Σ.

Let $can_{\underline{\hat{a}}}(\mathsf{newatoms}(\hat{c})) = \{\underline{n}_1, \ldots, \underline{n}_m\}$ and let the linear order $\hat{\prec}$ on $\hat{S}$ be a concatenation of the order $\prec$, restricted to $\mathsf{oldproven}(\hat{c})$, and the order $\underline{n}_1 \hat{\prec} \underline{n}_2 \hat{\prec} \cdots \hat{\prec} \underline{n}_m$. For each $1 \leqslant i \leqslant m$, Acheck generates a configuration $c_2^{(i)}$ defined as

$$c_2^{(i)} = (\, can_{\hat{a}}(\underline{\hat{a}}),\ can_{\hat{a}}(\hat{S}),\ can_{\hat{a}}(\mathsf{oldproven}(\hat{c})) \cup \{\underline{n}_1, \ldots, \underline{n}_{i-1}\},\ can_{\underline{\hat{a}}}(\hat{\prec}),\ \underline{n}_i).$$

This completes the description of the Acheck algorithm.

Theorem 5.22. *The* Acheck *algorithm is correct and runs in exponential time in case of bounded arities, and in double exponential time otherwise.*

Proof.

Soundness. It is easy to see that the algorithm is sound with respect to the standard chase, i.e., if $\mathsf{Acheck}(D, \Sigma, Q)$ returns "*true*", then $chase(D, \Sigma) \models Q$. In fact, modulo variable renaming, which preserves soundness according to Corollary 5.21, the algorithm does nothing but chasing D with respect to Σ, even if the chase steps are not necessarily in the same order as in the standard chase. Thus, each atom derived by Acheck occurs in *some* chase. Since every chase computes a universal solution that is complete with respect to conjunctive query answering, whenever Acheck returns "true", Q is entailed by some chase, and thus also by the standard chase, $chase(D, \Sigma)$.

Completeness. The completeness of Acheck with respect to $chase(D, \Sigma)$ can be shown as follows. Whenever $chase(D, \Sigma) \models Q$, there is a finite proof of Q, i.e., a finite sequence $proof_Q$ of generated atoms that ends with some atom $\underline{q}$, which is an instance of Q. This proof can be simulated by the alternating computation Acheck as follows: *(i)* steer the main branch of Acheck towards (a variant of) $\underline{q}$ by choosing successively the same TGDs and substitutions θ (modulo the appropriate variable renamings) as those used in the standard chase for the branch of $\underline{q}$; *(ii)* whenever a subcloud S has to be chosen for some atom $\underline{a}$ by Acheck, choose the set of atoms $cloud(D, \Sigma, \underline{a}) \cap (D \cup atoms(proof_Q))$, modulo appropriate variable renaming; *(iii)* for the ordering $\prec$, always choose the one given by $proof_Q$. The fact that no Q-instance is lost when replacing configurations by their canonical versions is guaranteed by Corollary 5.21.

Computational cost. In case of bounded arity, the size of each configuration c is polynomial in $D \cup \Sigma$. Thus, Acheck describes an alternating pspace (i.e., apspace) computation. It is well-known that apspace = exptime. In case the arity is not bounded, each configuration requires at most exponential space. The algorithm then describes a computation in Alternating expspace, which is equal to 2exptime. $\square$

Corollary 5.23. *Let Σ be a weakly guarded set of TGDs, and let D be a database over a schema $\mathcal{R}$. Then, computing $chase^{\perp}(D, \Sigma)$ can be done in exponential time in case of bounded arity, and in double exponential time otherwise.*

Proof. It is sufficient to start with an empty set A and then cycle over ground atoms $\underline{b}$ in the Herbrand base $HB(D)$ while checking whether $chase(D, \Sigma) \models \underline{b}$. If this holds, we add $\underline{b}$ to A. The result is $chase^{\perp}(D, \Sigma)$. The claimed time bounds follow straightforwardly. □

We can now finally state our independent proof of Theorem 5.9.

Proof of Theorem 5.9. We construct an algorithm Qcheck such that Qcheck(D, Σ, Q) outputs "true" iff $D \cup \Sigma \models Q$ (i.e., iff $chase(D, \Sigma) \models Q$). The algorithm relies on the notion of squid decompositions, and on Lemma 5.8; it works as follows.

(1) Qcheck starts by computing $chase^{\perp}(D, \Sigma)$.
(2) Qcheck nondeterministically guesses a squid decomposition $\delta = (Q^+, h, H, T)$ of Q based on a set $V_\delta \subseteq vars(h(Q^+))$, where $H = \{\underline{a} \in h(Q^+) \mid vars(\underline{a}) \subseteq V_\delta\}$ and T is $[V_\delta]$-acyclic. Additionally, Qcheck guesses a substitution $\theta_0 : V_\delta \rightarrow dom(D)$ such that $\theta_0(H) \subseteq chase^{\perp}(D, \Sigma)$. Note that this is an NP guess, because the number of atoms in Q^+ is at most twice the the number of atoms in Q.
(3) Qcheck checks whether θ_0 can be extended to a homomorphism θ such that $\theta(T) \subseteq chase^{+}(D, \Sigma)$. By Lemma 5.8, this is equivalent to check if $chase(D, \Sigma) \models Q$. Such a θ exists iff for each connected subgraph t of $\theta_0(T)$, there is a homomorphism θ_t such that $\theta_t(t) \subseteq chase^{+}(D, \Sigma)$. The Qcheck algorithm thus identifies the connected components of $\theta_0(T)$. Each such component is a $[dom(D)]$-acyclic conjunctive query, some of whose arguments may contain constants from $dom(D)$. Each such component can thus be represented as a $[dom(D)]$-join tree t. For each such join tree t, Qcheck now tests whether there exists a homomorphism θ_t such that $\theta_t(t) \subseteq chase^{+}(D, \Sigma)$. This is done by the subroutine Tcheck, that takes the TGD set Σ, the database D, and a connected subgraph (i.e., a subtree) t of $\theta_0(T)$ as input. The inner workings of Tcheck(D, Σ, t) are described below.
(4) Qcheck outputs "true" iff the above check *(3)* gives a positive result.

The correctness of Qcheck follows from Lemma 5.8. Given that step *(2)* is nondeterministic, the complexity of Qcheck is in NP^X, i.e., NP with an oracle in X, where X is a complexity class that is sufficiently powerful for: *(i)* computing $chase^{\perp}(D, \Sigma)$, and *(ii)* performing the tests Tcheck(D, Σ, t).

We now describe the Tcheck subroutine.

General notions. Tcheck(D, Σ, t) is obtained from Acheck via the following modifications. Each configuration of Tcheck maintains a pointer Tpoint to a vertex of t (an atom $\underline{a}_q$). Intuitively, this provides a link to the root of the subtree of t that still needs to be matched by descendant configurations of c. In addition to the data structures carried by each configuration of Acheck, each configuration of Tcheck also maintains an array subst of length w, where w is the maximum predicate arity in $\mathcal{R}$. Informally, subst encodes a substitution that maps the current atom of t to (the canonicalized version of) the current atom of $chase(D, \Sigma)$.

Tcheck works like Acheck, but instead of nondeterministically constructing a main configuration path of the configuration tree such that eventually some atom matches the query, it nondeterministically constructs a main configuration (sub)tree τ of the configuration tree, such that eventually all atoms of the join tree t get consistently translated into some vertices of τ. An important component of each main configuration c of Tcheck is its *current atom* $\underline{a}$. Initially, $\underline{a}$ is some nondeterministically chosen atom of D. For subsequent configurations of the alternating computation tree, $\underline{a}$ will take on nodes of $\mathrm{gcf}(D, \Sigma)$.

Initialization. Similarly to Acheck, the computation starts by generating an initial configuration $(\underline{a}, S, S, \prec, \star, \mathsf{Tpoint}, \mathsf{subst})$, where $\underline{a}$ is nondeterministically chosen from the database D, Tpoint points to the root $\underline{r}$ of t, and subst is a homomorphic substitution that $\mathsf{subst}(\underline{r}) = \underline{a}$, if $\underline{r}$ is homomorphically mappable on $\underline{a}$; otherwise subst is empty. This configuration will now be the root of the main configuration tree.

In general, the pointer Tpoint of each main configuration $c = (\underline{a}, S, S', \prec, \star, \mathsf{Tpoint}, \mathsf{subst})$ points to some atom $\underline{a}_q$ of t, which has not yet been matched. The algorithm attempts to expand this configuration by successively guessing a subtree of configurations, mimicking a suitable subtree of $\mathrm{gcf}(D, \Sigma)$ that satisfies the subquery of t rooted at $\underline{a}_q$.

Whenever Tcheck generates a further configuration, just as for Acheck, Tcheck generates via universal branching a number of configurations whose joint task is to verify that all elements of S are indeed provable. (We do not provide further details on how this branching is done.)

Expansion. The expansion of a main configuration $c = (\underline{a}, S, S', \prec, \star, \mathsf{Tpoint}, \mathsf{subst})$ works as follows. For a configuration c, Tcheck first checks whether there exists a homomorphism μ such that $\mu(\mathsf{subst}(\underline{a}_q)) = \underline{a}$.

1. *(μ exists.)* If μ exists, we have two cases:

   1.1. If $\underline{a}_q$ is a leaf of t, then the current configuration turns into an accepting one.

   1.2. If $\underline{a}_q$ is not a leaf of t, then Tcheck nondeterministically guesses whether μ is a *good match*, i.e., one that contributes to a global query answer and can be expanded to map the entire tree t into $\mathrm{gcf}(D, \Sigma)$.

      1.2.1. *(Good match).* In case of a good match, Tcheck branches universally and does the following *for each* child $\underline{a}_{qs}$ of $\underline{a}_q$ in t. It nondeterministically (i.e., via existential branching) creates a new configuration

$$c_s = can_{\underline{a}_s}(\underline{a}_s, S_s, S'_s, \prec_s, \star, \mathsf{Tpoint}_s, \mathsf{subst}_s)$$

      where Tpoint_s points to $\underline{a}_{qs}$, and where subst_s encodes the composition $\mu \circ \mathsf{subst}_s$. The atom $\underline{a}_s$ is guessed, analogously to what is done in Acheck, by guessing some TGD $\rho \in \Sigma$ having body Φ and head atom $\underline{\psi}$, such that the guard atom $\underline{g}$ matches $\underline{a}$ via some homomorphism θ (that is, $\theta(\underline{g}) = \underline{a}$) and where $\theta(\Phi) \subseteq S'$. The cloud subsets S_s and S'_s are chosen again as in Acheck. Intuitively, here Tcheck, having found a good match of $\underline{a}_q$ on $\underline{a}$, tries to match the children of $\underline{a}_q$ in t to children (and, eventually, descendants) of $\underline{a}$ in $\mathrm{gcf}(D, \Sigma)$. Finally, the function $can_{\underline{a}_s}$ indicates that appropriate canonizations are made to obtain c_s from c (we omit the tedious details).

1.2.2. *(No good match).* In case no good match exists, a child configuration

$$c_{new} = can_{\underline{a}_{new}}(\underline{a}_{new}, S, S', \prec, \star, \mathsf{Tpoint}, \mathsf{subst})$$

of c is nondeterministically created, whose first component represents a child $\underline{a}_{new}$ of $\underline{a}$, and where c_{new} inherits all of its remaining components from c. Intuitively, after having failed at matching $\underline{a}_q$ (to which, we remind, Tpoint points) to $\underline{a}$, Tcheck attempts at matching the same $\underline{a}_q$ to some child of $\underline{a}$ in $\mathrm{gcf}(D, \Sigma)$. By analogy with the previous case, a_{new} is obtained by guessing some TGD $\rho \in \Sigma$ having body Φ and head atom $\underline{\psi}$, such that the guard atom $\underline{g}$ matches $\underline{a}$ via some homomorphism θ (that is, $\theta(\underline{g}) = \underline{a}$), $\theta(\Phi) \subseteq S'$, and where $\underline{a}_{new} := \theta(\underline{\psi})$. Again, the function term $can_{\underline{a}_{new}}$ indicates that appropriate canonizations are applied (which we do not describe in detail).

2. *(μ does not exist.)* In this case, Tcheck proceeds exactly as in case 1.2.2, namely, it attempts at matching the same $\underline{a}_q$ to some child (or eventually some descendant) of $\underline{a}$ in $\mathrm{gcf}(D, \Sigma)$.

Correctness. The correctness of Tcheck can be shown along similar lines as for Acheck. An important additional point to consider for Tcheck is that, given that the query t is acyclic, it is actually sufficient to remember at each configuration c only the latest "atom" substitution subst. The correctness of Qcheck then follows from the correctness of Tcheck and from Lemma 5.8.

Computational cost. As for the complexity of Qcheck, note that in case the arity is bounded, Tcheck runs in APSPACE = EXPTIME, and computing $chase^{\perp}(D, \Sigma)$ is in EXPTIME by Corollary 5.23. Thus, Qcheck runs in time $\mathrm{NP}^{\mathrm{EXPTIME}} = \mathrm{EXPTIME}$. In case of unbounded arities, both computing $chase^{\perp}(D, \Sigma)$ and running Tcheck are in 2EXPTIME, therefore Qcheck runs in time $\mathrm{NP}^{2\mathrm{EXPTIME}} = 2\mathrm{EXPTIME}$. □

By combining Theorems 4.1 and 5.9 we immediately get the following characterization for the complexity of reasoning under weakly guarded sets of TGDs.

Theorem 5.24. *Let Σ be a weakly guarded set of TGDs on a schema $\mathcal{R}$, D a database for $\mathcal{R}$, and Q a Boolean conjunctive query. Determining whether $D \cup \Sigma \models Q$ or, equivalently, whether $chase(D, \Sigma) \models Q$ is* EXPTIME*-complete in case of bounded predicate arities, even if Σ is fixed and Q is atomic. In the general case of unbounded predicate arities, the same problem is* 2EXPTIME*-complete. The same completeness results hold for the problem of query containment under weakly guarded sets of TGDs.*

Generalization. The definition of WGTGDs can be generalized to classes of TGDs whose unguarded positions are guaranteed to contain a controlled finite number of null-values only. Let f be a computable integer function in two variables. Call a predicate position π of a TGD set Σ *f-bounded* if no more than $f(|D|, |\Sigma|)$ null values appear in $chase(D, \Sigma)$ as arguments in position π; otherwise call Σ *f-unbounded*. A set Σ of TGDs is *f-weakly guarded* if each each rule of Σ contains an atom in its body that covers all variables which occur within this rule in f-unbounded positions only. By a very minor adaptation of the proof of Theorem 3.14, it can be seen that CQ-answering for the class of f-weakly guarded TGDs is decidable. Moreover, by a simple modification of the Qcheck and Tcheck procedures, allowing a polynomial number of nulls to enter "unguarded" positions, it can be shown that CQ-answering for fixed sets Σ of W*GTGDs is EXPTIME-complete in the worst case, where the class of W*GTGD sets is defined as follows. A set Σ of TGDs belongs to this class if Σ is f-weakly guarded for some function f for which there exists a function g, such that $f(|D|, |\Sigma|)| \leqslant |D|^{g(|\Sigma|)}$.

6. Guarded TGDs

We now turn our attention to GTGDs. We first consider the case of a variable database D as input. Later, we prove part of the complexity bounds under the stronger condition of fixed database.

6.1 Complexity—Variable Database

Theorem 6.1. *Let Σ be a set of GTGDs over a schema $\mathcal{R}$ and D be a database for $\mathcal{R}$. Let, as before, w denote the maximum predicate arity in $\mathcal{R}$ and $|\mathcal{R}|$ the total number of predicate symbols in $\mathcal{R}$. Then:*

(1) Computing $chase^{\perp}(D, \Sigma)$ can be done in polynomial time if both w and $|\mathcal{R}|$ are bounded and, thus, also in case of a fixed set Σ. The same problem is in EXPTIME *(and thus* EXPTIME*-complete) if w is bounded, and in* 2EXPTIME *otherwise.*
(2) If Q is an atomic or fixed Boolean query then checking whether $chase(D, \Sigma) \models Q$ is PTIME*-complete when both w and $|\mathcal{R}|$ are bounded. The same problem remains* PTIME*-complete even in case Σ is fixed. This problem is* EXPTIME*-complete if w is bounded and* 2EXPTIME*-complete in general. It remains* 2EXPTIME*-complete even when $|\mathcal{R}|$ is bounded.*
(3) If Q is a general conjunctive query, checking $chase(D, \Sigma) \models Q$ is NP*-complete in case both w and $|\mathcal{R}|$ are bounded and, thus, also in case of a fixed set Σ. Checking $chase(D, \Sigma) \models Q$ is* EXPTIME*-complete if w is bounded and* 2EXPTIME*-complete in general. It remains* 2EXPTIME*-complete even when $|\mathcal{R}|$ is bounded.*
(4) BCQ answering under GTGDs is NP*-complete if both w and $|\mathcal{R}|$ are bounded, even in case the set Σ of GTGDs is fixed.*
(5) BCQ answering under GTGDs is EXPTIME*-complete if w is bounded and* 2EXPTIME*-complete in general. It remains* 2EXPTIME*-complete even when $|\mathcal{R}|$ is bounded.*

Proof. First, note that items *(4)* and *(5)* immediately follow from the first three items, given that $chase(D, \Sigma)$ is a universal model. We therefore just need to prove items *(1)*-*(3)*. We first explain how the hardness results are obtained, and then deal with the matching membership results.

Hardness Results. The PTIME-hardness of checking $chase(D, \Sigma) \models Q$ for atomic (and thus also fixed) queries Q and for fixed Σ follows from the fact that ground atom inference from a fixed fully guarded Datalog program over variable databases is PTIME-hard. In fact, in the proof of Theorem 4.4 in the work by Dantsin, Eiter, Gottlob, and Voronkov (2001) it is shown that fact inference from a single-rule Datalog program whose body has a guard atom that contains all variables is PTIME-hard. The NP-hardness in item *(3)* is immediately derived from the hardness of CQ containment (which in turn is polynomially equivalent to query answering) without constraints (Chandra & Merlin, 1977). The hardness results for EXPTIME and 2EXPTIME are all derived via minor variations of the proof of Theorem 4.1. For example, when $|\mathcal{R}|$ is unbounded and w is bounded, the tape cells of the polynomial worktape are simulated by using polynomially many predicate symbols. For example, the fact that in configuration v cell 5 contains symbol 1 can be encoded as $S_5^1(v)$. We omit further details, given that a much stronger hardness result will be established via a full proof in Theorem 6.2.

MEMBERSHIP RESULTS. The membership results are proved exactly as those for weakly guarded sets of TGDs, except that instead of using the concept of cloud, we now use a similar concept of *restricted cloud*, which coincides with that of a *type* of an atom in the work by Calì et al. (2012a). The restricted cloud $rcloud(D, \Sigma, \underline{a})$ of an atom $\underline{a} \in chase(D, \Sigma)$ is the set of all atoms $\underline{b} \in chase(D, \Sigma)$ such that $dom(\underline{b}) \subseteq dom(\underline{a})$. By a proof that is almost identical to the one of Theorem 5.16, we can show that if D is a database, Σ a set of GTGDs, and if $\underline{a} \in chase(D, \Sigma)$, then $\nabla^r \underline{a} = \mathrm{gcf}[\underline{a}, rcloud(D, \Sigma, \underline{a})]$, where $\nabla^r \underline{a}$ is defined as $\nabla^r \underline{a} = \{\underline{a}_{\downarrow}\} \cup rcloud(D, \Sigma, \underline{a})$. It follows that, for the main computational tasks, we can use algorithms rAcheck, rQcheck, and rTcheck, which differ from the already familiar Acheck, Qcheck, and Tcheck only in that restricted clouds instead of the ordinary clouds are used. However, unlike the case when both $|\mathcal{R}|$ and w are bounded and a cloud (or subcloud) can have polynomial size in $|D \cup \Sigma|$, a restricted cloud $rcloud(D, \Sigma, \underline{a})$ has a constant number of atoms, and storing its canonical version $can_{\underline{a}}(rcloud(D, \Sigma, \underline{a}))$ thus requires logarithmic space only. In total, in case both $|\mathcal{R}|$ and w are bounded, due to the use of restricted clouds (and subsets thereof) each configuration c of rAcheck and of rTcheck only requires logarithmic space. Since ALOGSPACE = PTIME, the PTIME-results for atomic queries in items *(1)* and *(2)* follow. Moreover, if both $|\mathcal{R}|$ and w are bounded, for general (non-atomic and non-fixed) queries, the rQcheck algorithm decides if $chase(D, \Sigma) \models Q$ in NP by guessing a squid decomposition (in nondeterministic polynomial time) and checking (in ALOGSPACE=PTIME) if there is a homomorphism from this squid decomposition into $chase(D, \Sigma)$. Thus, in this case, rQcheck runs in $\mathrm{NP}^{\mathrm{PTIME}} = \mathrm{NP}$, which proves the NP upper bound of Item *(3)*. If, in addition, Q is fixed, then Q has only a constant number of squid decompositions, and therefore rQcheck runs in $\mathrm{PTIME}^{\mathrm{PTIME}} = \mathrm{PTIME}$, which proves the PTIME upper bound for fixed queries mentioned in item *(2)*. The EXPTIME and 2EXPTIME upper bounds are inherited from the same upper bounds for WGTGDs. □

Note that one of the main results by Johnson and Klug (1984), namely, that query containment under inclusion dependencies of bounded arities is NP-complete, is a special case of Item *(3)* of Theorem 6.1.

6.2 Complexity—Fixed Database

The next result tightens parts of Theorem 6.1 by showing that the above EXPTIME and 2EXPTIME-completeness results hold even in case of a fixed input database.

Theorem 6.2. *Let Σ be set of GTGDs on a schema $\mathcal{R}$. As before, let w denote the maximum arity of predicate in $\mathcal{R}$ and $|\mathcal{R}|$ be the total number of predicate symbols. Then, for fixed databases D, checking whether $chase(D, \Sigma) \models Q$ is* EXPTIME*-complete if w is* *bounded and* 2EXPTIME-*complete for unbounded* w*. For unbounded* w*, this problem remains* 2EXPTIME-*complete even when* $|\mathcal{R}|$ *is bounded.*

Proof. First, observe that the upper bounds (i.e., the membership results for EXPTIME and 2EXPTIME) are inherited from Theorem 6.1, so it suffices to prove the hardness results for the cases where Q is a fixed atomic query.

We start by proving that checking $chase(D, \Sigma) \models Q$ is EXPTIME-hard if w is bounded. It is well-known that APSPACE (alternating PSPACE) equals EXPTIME.

As already noted in the proof of Theorem 4.1, it is sufficient to simulate an LINSPACE alternating Turing machine (ATM) $\mathcal{M}$ that uses at most n worktape cells on every input (bit string) I of size n, where the input string is initially present on the worktape. In particular, we will show that $\mathcal{M}$ accepts the input I iff $chase(D, \Sigma) \models Q$.

Without loss of generality, we assume that *(i)* ATM $\mathcal{M}$ has exactly one accepting state, a, which is also a halting state; *(ii)* the initial state of $\mathcal{M}$ is an existential state; *(iii)* $\mathcal{M}$ alternates at each transition between existential and universal states; and *(iv)* $\mathcal{M}$ never tries to read beyond its tape boundaries.

Let $\mathcal{M}$ be defined as $\mathcal{M} = (S, \Lambda, \delta, q_0, \{s_a\})$, where S is the set of states, $\Lambda = \{0, 1, \flat\}$ is the tape alphabet, $\flat \in \Lambda$ is the blank tape symbol, $\delta : S \times \Lambda \to (S \times \Lambda \times \{\ell, r, \bot\})^2$ is the transition function ($\bot$ denotes the "stay" head move, while ℓ and r denote "left" and "right" respectively), $q_0 \in S$ is the initial state, and $\{s_a\}$ is the singleton set of final (accepting) states. Since $\mathcal{M}$ is an alternating TM, its set of states S is *partitioned* into two sets, $S_\forall$ and $S_\exists$—universal and existential states, respectively. The general idea of the encoding is that the different *configurations* of $\mathcal{M}$ on input I of length n will be represented by fresh nulls that are generated in the construction of the chase.

Let us now describe the schema $\mathcal{R}$. First, for each integer $1 \leqslant i \leqslant n$, $\mathcal{R}$ contains the predicate $head_i/1$, such that $head_i(c)$ be true iff at configuration c the head of $\mathcal{M}$ is over the tape cell i. $\mathcal{R}$ also has the predicates $zero_i/1$, $one_i/1$, and $blank_i/1$, where $zero_i(c)$, $one_i(c)$, and $blank_i(c)$ are true if in configuration c the tape cell i contains the symbol 0, 1, or $\flat$, respectively. Furthermore, for each state $s \in S$, $\mathcal{R}$ has a predicate $state_s/1$, such that $state_s(c)$ is true iff the state of configuration c is s. $\mathcal{R}$ also contains: the predicate $start/1$, where $start(c)$ is true iff c is the starting configuration; the predicate $config/1$, which is true iff its argument identifies a configuration; and the predicate $next/3$, where $next(c, c_1, c_2)$ is true if c_1 and c_2 are the two successor configurations of c. There are also predicates $universal/1$ and $existential/1$, such that $universal(c)$ and $existential(c)$ are true if c is a universal (respectively, existential) configuration. Finally, there is a predicate $accepting/1$, where $accepting(c)$ is true only for accepting configurations c, and a propositional symbol $accept$, which is true iff the Turing Machine $\mathcal{M}$ accepts the input I.

We now describe a set $\Sigma(\mathcal{M}, I)$ of GTGDs that simulates the behavior of $\mathcal{M}$ on input I. The rules of $\Sigma(\mathcal{M}, I)$ are as follows.

1. *Initial configuration generation rules.* The following rule creates an initial state: $\to \exists X\, init(X)$. We also add a rule $init(X) \to config(X)$, which says that the initial configuration is, in fact, a configuration.
2. *Initial configuration rules.* The following set of rules encodes the tape content of the initial configuration, that is, the input string I. For each $1 \leqslant i \leqslant n$, if the i-th cell of the tape contains a 0, we add the rule $init(X) \to zero_i(X)$; if it contains a 1, we

add $\mathit{init}(X) \rightarrow \mathit{one}_i(X)$. We also add the rule $\mathit{init}(X) \rightarrow \mathit{existential}(X)$ in order to say, without loss of generality, that the initial configuration is an existential one. Moreover, we add the rules $\mathit{init}(X) \rightarrow \mathit{head}_1(X)$ and $\mathit{init}(X) \rightarrow \mathit{state}_{s_0}(X)$ to define the initial values of the state and the head position of $\mathcal{M}$ on input I.

3. *Configuration generation rules.* We add a rule that creates two successor configuration identifiers for each configuration identifier. Moreover, we add rules stating that these new configuration identifiers indeed identify configurations:

$$\begin{array}{rcl} \mathit{config}(X) & \rightarrow & \exists X_1, X_2\, \mathit{next}(X, X_1, X_2), \\ \mathit{next}(X, Y, Z) & \rightarrow & \mathit{config}(Y), \\ \mathit{next}(X, Y, Z) & \rightarrow & \mathit{config}(Z). \end{array}$$

4. *Transition rules.* We show by example how transition rules are generated for each transition in the finite control. Assume, for instance, that the transition table contains a specific transition of the form: $(s, 0) \rightarrow (\,(s1, 1, r)\,,\,(s2, 0, \ell)\,)$. Then we assert the following rules, for $1 \leqslant i \leqslant n$:

$$\begin{array}{rcl} \mathit{head}_i(X), \mathit{zero}_i(X), \mathit{state}_s(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{state}_{s_1}(X_1) \\ \mathit{head}_i(X), \mathit{zero}_i(X), \mathit{state}_s(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{state}_{s_2}(X_2). \end{array}$$

Moreover, for each $1 \leqslant i < n$ we have these rules:

$$\begin{array}{rcl} \mathit{head}_i(X), \mathit{zero}_i(X), \mathit{state}_s(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{one}_i(X_1) \\ \mathit{head}_i(X), \mathit{zero}_i(X), \mathit{state}_s(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{head}_{i+1}(X_1), \end{array}$$

and for each $1 < i \leqslant n$ we add these rules:

$$\begin{array}{rcl} \mathit{head}_i(X), \mathit{zero}_i(X), \mathit{state}_s(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{zero}_i(X_2) \\ \mathit{head}_i(X), \mathit{zero}_i(X), \mathit{state}_s(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{head}_{i-1}(X_2) \end{array}$$

The other types of transition rules are constructed analogously. Note that the total number of rules added is $6n$ times the number of transition rules. Hence it is linearly bounded by the size n of the input string I to $\mathcal{M}$.

5. *Inertia rules.* These rules state that tape cells in positions not under the head keep their values. Thus, for each $1 \leqslant i, j \leqslant n$ such that $i \neq j$ we add the rules:

$$\begin{array}{rcl} \mathit{head}_i(X), \mathit{zero}_j(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{zero}_j(X_1) \\ \mathit{head}_i(X), \mathit{one}_j(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{one}_j(X_1) \\ \mathit{head}_i(X), \mathit{blank}_j(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{blank}_j(X_1), \end{array}$$

6. *Configuration-type rules.* These rules say that the immediate successor configurations of an existential configuration are universal, and vice-versa:

$$\begin{array}{rcl} \mathit{existential}(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{universal}(X_1) \\ \mathit{existential}(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{universal}(X_2) \\ \mathit{universal}(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{existential}(X_1) \\ \mathit{universal}(X), \mathit{next}(X, X_1, X_2) & \rightarrow & \mathit{existential}(X_2). \end{array}$$

7. *Acceptance rules.* These recursive rules state when a configuration is accepting:

$$
\begin{array}{rcl}
\mathit{state}_{s_a}(X) & \rightarrow & \mathit{accepting}(X) \\
\mathit{existential}(X), \mathit{next}(X, X_1, X_2), \mathit{accepting}(X_1) & \rightarrow & \mathit{accepting}(X) \\
\mathit{existential}(X), \mathit{next}(X, X_1, X_2), \mathit{accepting}(X_2) & \rightarrow & \mathit{accepting}(X) \\
\mathit{universal}(X), \mathit{next}(X, X_1, X_2), \mathit{accepting}(X_1), \mathit{accepting}(X_2) & \rightarrow & \mathit{accepting}(X) \\
\mathit{init}(X), \mathit{accepting}(X) & \rightarrow & \mathit{accept}.
\end{array}
$$

This completes the description of the set of TGDs $\Sigma(\mathcal{M}, I)$. Note that this set is guarded, has maximum predicate arity 3, can be obtained in logarithmic space from I and the constant machine description of $\mathcal{M}$. It faithfully simulates the behavior of the alternating linear space machine $\mathcal{M}$ in input I. It follows that $\Sigma(\mathcal{M}, I) \models \mathit{accept}$ iff $\mathcal{M}$ accepts input I. Let D_0 denote the empty database, and let Q_0 be the ground-atom query *accept*. We then have that $\Sigma(\mathcal{M}, I) \cup D_0 \models Q_0$ iff $\mathcal{M}$ accepts input I. This shows that answering ground atom queries on fixed databases constrained by bounded arity GTGDs is EXPTIME-hard.

Let us now illustrate how we obtain the 2EXPTIME hardness result for guarded TGDs when arities are unbounded, but when the number $|\mathcal{R}|$ of predicate symbols of the schema $\mathcal{R}$ is bounded by a constant. Given that AEXPSPACE=2EXPTIME (AEXPSPACE $\equiv$ alternating AEXPSPACE), our aim is now to simulate an AEXPSPACE Turing machine. It is sufficient to simulate one that uses no more than 2^n worktape cells, since the acceptance problem for such machines is already 2EXPTIME-hard. In fact, by trivial padding arguments, the acceptance problem for every AEXPSPACE machine can be transformed in polynomial time into the acceptance problem for one using at most 2^n worktape cells.

The problem is, however, that now we can no longer construct a polynomial number of rules that explicitly address each worktape cell i, or each pair of cells i, j, since now there is an *exponential* number of worktape cells. The idea now is to encode tape cell indexes as *vectors* of symbols $(v_1, \ldots, v_k)$ where $v_i \in \{0, 1\}$. As in the proof of Theorem 4.1, we could define, with a polynomial number of rules, a successor relation *succ* that stores pairs of indexes as $\mathit{succ}(v_1, \ldots, v_k, w_1, \ldots, w_k)$. However, there is a further difficulty: we now have two different types of variables: the variables V_i, W_j that range over the bits v_i, w_i in the above-described bit vectors, and the variables X, Y, Z that range over configurations. A major difficulty is that, given that our rules are all guarded, we must make sure that these two types of variables, whenever they occur elsewhere in a rule body, also occur in some guard. To this end, we will use a fixed database D_{01} that contains the single fact $\mathit{zeroone}(0, 1)$, and we will construct a "guard" relation g such that for each vector $\mathbf{v}$ of n bits and its binary successor $\mathbf{w}$, and for each configuration x with its two successor configurations y and z, the relation g contains a tuple $g(\mathbf{v}, \mathbf{w}, x, y, z)$. We will use several auxiliary relations to construct g.

For technical reasons, the first two arguments of some atoms below will be dummy variables T_0 and T_1 that will always be forced to take the values 0 and 1, respectively. This way, where convenient, we will have the values 0 and 1 available implicitly in form of variables, and we will not need to use these constants explicitly in our rules.

Given that our database is now non-empty, we do not need to create the initial configuration identifier via an existential rule as before. We can simply take 0 as the identifier of this initial configuration: $\mathit{zeroone}(T_0, T_1) \rightarrow \mathit{init}(T_0, T_1, T_0)$. (Here, the first two arguments of $\mathit{init}(T_0, T_1, T_0)$ just serve, as explained, to carry the values 0 and 1 along.) We also add: $\mathit{init}(T_0, T_1, T_0) \rightarrow \mathit{config}(T_0, T_1, T_0)$ to assert that 0 is the identifier of the initial configuration. Next we present the new configuration generation rules.

$$\begin{array}{rcl} \mathit{config}(T_0, T_1, X) & \rightarrow & \exists Y, \exists Z\, \mathit{next}(T_0, T_1, X, Y, Z), \\ \mathit{next}(T_0, T_1, X, Y, Z) & \rightarrow & \mathit{config}(T_0, T_1, Y), \\ \mathit{next}(T_0, T_1, X, Y, Z) & \rightarrow & \mathit{config}(T_0, T_1, Z). \end{array}$$

We use further rules to create a relation b such that each atom $b(0, 1, \mathbf{v}, x, y, z)$ contains a tuple for each vector $\mathbf{v}$ of n bits, and for each configuration x. For better readability, whenever useful, we will use superscripts for indicating the arity of vector variables: for instance, $\mathbf{V}^{(n)}$ denotes $V_1, \ldots, V_n$. Moreover, $\mathbf{0}^{(j)}$ denotes the vector of j zeros and $\mathbf{1}^{(j)}$ the vector of j ones. We start with the rule $\mathit{next}(T_0, T_1, X, Y, Z) \rightarrow b(T_0, T_1, \mathbf{T_0}^{(n)}, X, Y, Z)$, which defines an atom $b(0, 1, \mathbf{0}^{(n)}, x, y, z)$, for each configuration x and its next-successors y and z.

The following n rules, for $1 \leqslant i \leqslant n$, generate an exponential number of new atoms, for each triple X, Y, Z, by swapping 0s to 1s in all possible ways. Eventually, the chase will generate all possible prefixes of n bits.

$$\begin{array}{l} b(T_0, T_1, U_1, \ldots, U_{i-1}, T_0, U_{i+1}, \ldots, U_n, X, Y, Z) \rightarrow \\ \qquad\qquad\qquad b(T_0, T_1, U_1, \ldots, U_{i-1}, T_1, U_{i+1}, \ldots, U_n, X, Y, Z). \end{array}$$

We are now ready to define the guard-relation g through another group of guarded rules. For each $0 \leqslant r < n$, we add:

$$b(T_0, T_1, \mathbf{U}^{(r)}, T_0, \mathbf{T_1}^{(n-r-1)}, X, Y, Z) \rightarrow g(\mathbf{U}^{(r)}, T_0, \mathbf{T_1}^{(n-r-1)}, \mathbf{U}^{(r)}, T_1, \mathbf{T_0}^{(n-r-1)}, X, Y, Z).$$

The above n rules define an exponential number of cell-successor pairs for each triple of configuration identifiers X, Y, Z, where Y and Z are the "next" configurations following X. In particular, the relation g contains precisely all tuples $g(\mathbf{v}, \mathbf{w}, x, y, z)$, such that $\mathbf{v}$ is an n-ary bit vector, $\mathbf{w}$ is its binary successor, x is a configuration identifier, y is its first successor via the next relation, and z is its second successor via the next relation.

We are now ready to simulate an aexpspace Turing machine $\mathcal{M}'$ over an input string I by a set of GTGDs $\Sigma(\mathcal{M}', I)$. Since this simulation is similar to the one presented in the first part of this proof, we just sketch it and point out the main differences.

For the simulation, we use (in addition to the aforementioned auxiliary predicates) predicates similar to the ones used earlier for the simulation of the exptime Turing machine $\mathcal{M}$. However, we only use a constant number of predicates. So, rather than using, atoms $\mathit{head}_i(x)$, $\mathit{zero}_i(x)$ and so on, we use their vectorized versions $\mathit{head}(\mathbf{v}, x)$, $\mathit{zero}(\mathbf{v}, x)$ and so on, where $\mathbf{v}$ is a bit vector of length n that takes the role of an exponential index. Thus, for example, the equivalent of the earlier rule

$$\mathit{head}_i(X), \mathit{zero}_i(X), \mathit{state}_s(X), \mathit{next}(X, X_1, X_2) \rightarrow \mathit{one}_i(X_1)$$

is $g(\mathbf{V}, \mathbf{W}, X, X_1, X_2), \mathit{head}(\mathbf{V}, X), \mathit{zero}(\mathbf{V}, X), \mathit{state}(X, s) \rightarrow \mathit{one}(\mathbf{V}, X_1)$. The earlier rule

$$\mathit{head}_i(X), \mathit{zero}_i(X), \mathit{state}_s(X), \mathit{next}(X, X_1, X_2) \rightarrow \mathit{head}_{i-1}(X_2)$$

becomes $g(\mathbf{V}, \mathbf{W}, X, X_1, X_2), \mathit{head}(\mathbf{W}, X), \mathit{zero}(\mathbf{W}, X), \mathit{state}(X, s) \rightarrow \mathit{head}(\mathbf{V}, X_2)$. It is now straightforward to see how the initialization rules can be written. Informally, for copying the input string I to the worktape, we place the n input bits of I on the tape by writing a rule for each such bit. We then add rules that fill all positions from $n+1$ to 2^n with blanks. As this can be done in a similar way as in the second part of the proof of Theorem 4.1, we omit the details.

The only remaining issue is the specification of the inertia rules. These rules deal with *pairs* i, j of different, not necessary adjacent, tape cell positions in our earlier simulation. Here we have only adjacent cell positions available so far. The problem can be solved in different ways. One possibility is described below.

We can simply modify the definition of the predicate b by adding a second vector of n bits to the b-atoms so that b-atoms actually have the form $b(T_0, T_1, \mathbf{v}, \mathbf{u}, x, y, z)$, where $\mathbf{v}$ and $\mathbf{u}$ range over all possible *distinct pairs* of bit vectors of length n. This $\mathbf{u}$ vector is then carried over to the g-atoms. We can thus assume that the g-atoms now have the form $g(\mathbf{v}, \mathbf{w}, \mathbf{u}, x, y, z)$. The former inertia rule $\mathit{head}_i(X), \mathit{zero}_j(X), \mathit{next}(X, X_1, X_2) \rightarrow \mathit{zero}_j(X_1)$ would then become $g(\mathbf{V}, \mathbf{W}, \mathbf{U}, X, X_1, X_2), \mathit{head}(\mathbf{W}, X), \mathit{zero}(\mathbf{U}, X) \rightarrow \mathit{zero}(\mathbf{U}, X_1)$.

What remains to be defined are the configuration and the acceptance rules. The configuration rules are very similar to the ones used in the previous reduction, hence we leave them as an exercise. The acceptance rules are as follows:

$$\begin{array}{rcl}
\mathit{state}(X, s_a) & \rightarrow & \mathit{accepting}(X) \\
\mathit{existential}(X), g(\mathbf{V}, \mathbf{W}, X, X_1, X_2), \mathit{accepting}(X_1) & \rightarrow & \mathit{accepting}(X) \\
\mathit{existential}(X), g(\mathbf{V}, \mathbf{W}, X, X_1, X_2), \mathit{accepting}(X_2) & \rightarrow & \mathit{accepting}(X) \\
\mathit{universal}(X), g(\mathbf{V}, \mathbf{W}, X, X_1, X_2), \mathit{accepting}(X_1), \mathit{accepting}(X_2) & \rightarrow & \mathit{accepting}(X) \\
\mathit{zeroone}(T_0, T_1), \mathit{accepting}(T_0) & \rightarrow & \mathit{accept}.
\end{array}$$

This completes the description of the set of TGDs $\Sigma(\mathcal{M}', I)$. Note that this set is guarded and has a constant number of predicates. It can be obtained in LOGSPACE from I and the constant machine description of $\mathcal{M}$. It also faithfully simulates the behavior of the alternating exponential space machine $\mathcal{M}'$ on input I. It follows that $\Sigma(\mathcal{M}', I) \models \mathit{accept}$ iff $\mathcal{M}'$ accepts input I. Let Q_0 be the BCQ defined as $Q_0 = \{\mathit{accept}\}$. We then have $D_{01} \cup \Sigma(\mathcal{M}', I) \models Q_0$ iff $\mathcal{M}'$ accepts input I. This shows that answering ground atomic queries on fixed databases under guarded TGDs with a fixed number of predicate symbols (but unbounded arity) is 2EXPTIME-hard. □

7. Polynomial Clouds Criterion

In the previous section we have seen that, in case of bounded arity, query answering under weakly guarded sets of TGDs is EXPTIME-complete, while query answering under GTGDs is NP-complete. Note that, for unrestricted queries and databases, NP-completeness is the best we can obtain. In fact, even in the absence of constraints, the BCQ answering problem is NP-complete (Chandra & Merlin, 1977).

In this section, we establish a criterion that can be used as a tool for recognizing relevant cases where query answering is in NP even for weakly guarded sets of TGDs that are not fully guarded. Note that we consider both a setting where the weakly guarded set Σ of

TGDs is fixed and a setting where *classes* of TGD sets are considered. For these classes, we require *uniform* polynomial bounds.

Definition 7.1. [Polynomial Clouds Criterion] A fixed weakly guarded set Σ of TGDs satisfies the *Polynomial Clouds Criterion* (*PCC*) if both of the following conditions hold:

1. There exists a polynomial $\pi(\cdot)$ such that for each database D, $|clouds(D,\Sigma)/_{\simeq}| \leqslant \pi(|D|)$. In other words, up to an isomorphism, there are only polynomially many clouds.
2. There is a polynomial $\pi'(\cdot)$ such that, for each database D and for each atom $\underline{a}$:
   - if $\underline{a} \in D$ then $cloud(D,\Sigma,\underline{a})$ can be computed in time $\pi'(|D|,|\Sigma|)$, and
   - if $\underline{a} \notin D$ then $cloud(D,\Sigma,\underline{a})$ can be computed in time $\pi'(|D|,|\Sigma|)$ starting with D, $\underline{a}$, and $cloud(D,\Sigma,\underline{b})$, where $\underline{b}$ is the predecessor of $\underline{a}$ in $\mathrm{gcf}(D,\Sigma)$.

   We also say that Σ satisfies the PCC with respect to π and π'. Note that in the above, $|\Sigma|$ is constant and can be omitted. However, the use of $|\Sigma|$ is justified by the following. A class $\mathcal{C}$ of weakly guarded TGD sets satisfies the PCC if there are *fixed* polynomials π and π' such that each TGD set in $\mathcal{C}$ satisfies the PCC *uniformly* with respect to π and π' (i.e., each TGD set in this class has π, π' as a bound).

Theorem 7.2. *Let Σ be a fixed weakly guarded set of TGDs over a schema $\mathcal{R}$, such that Σ enjoys the Polynomial Clouds Criterion. Then:*

- *Deciding for a database D and an atomic or fixed Boolean conjunctive query Q whether $D \cup \Sigma \models Q$ (equivalently, whether $chase(D,\Sigma) \models Q$) is in* PTIME*.*
- *Deciding for a database D and a general Boolean conjunctive query Q whether $D \cup \Sigma \models Q$ (equivalently, $chase(D,\Sigma) \models Q$) is in* NP*.*

Proof. A polynomial algorithm Acheck2 for atomic queries Q works as follows. We start to produce the chase forest $\mathrm{gcf}(D,\Sigma)$ using the standard chase. In addition, immediately after generating any node $\underline{a}$ and its cloud $cloud(D,\Sigma,\underline{a})$ (in polynomial time), we will store $can_{\underline{a}}(\underline{a}, cloud(D,\Sigma,\underline{a}))$ in a buffer, which we call *cloud-store*. Whenever a branch of the forest reaches a vertex $\underline{b}$ such that $can_{\underline{b}}(cloud(D,\Sigma,\underline{b}))$ is already in the cloud-store, further expansion of that branch $\underline{b}$ is blocked. Since there can be only a polynomial number of pairs $can_{\underline{a}}(\underline{a}, cloud(D,\Sigma,\underline{a}))$, the algorithm stops after a polynomial number of chase steps, each step requiring only polynomial time. Now, by Corollary 5.17, the cloud-store already contains all possible atoms of $chase(D,\Sigma)$ and their clouds, up to isomorphism. To check whether $chase(D,\Sigma) \models Q$ holds for an atomic query Q, it is thus sufficient to test whether every atom $\underline{c}$ that occurs in the cloud-store matches Q. In summary, Acheck2 runs in PTIME.

The algorithm Qcheck2 for conjunctive queries works just like Qcheck, except that it calls the algorithm Tcheck2 as a subroutine instead of Tcheck. The input to Tcheck2 is D, Q, and also the cloud-store computed by Acheck2. We further assume that this cloud-store identifies each entry $e = can_{\underline{a}}(\underline{a}, cloud(D,\Sigma,\underline{a}))$ by a unique integer $e_{\#}$ using $O(\log n)$ bits only. Tcheck2 is an alternating algorithm that works essentially like Tcheck, except for the following modifications:

- Tcheck always guesses the full cloud $S = cloud(D,\Sigma,\underline{a})$, instead of possibly guessing a subcloud. In contrast, Tcheck2 just guesses the entry number $e_{\#}$ of the corresponding entry $can_{\underline{a}}(\underline{a}, cloud(D,\Sigma,\underline{a}))$ of the cloud-store.

- $\mathsf{Tcheck2}$ verifies correctness of the cloud guess in ALOGSPACE using D, $\underline{a}$, $e_{\#}$, as well as $\underline{b}$ and $e'_{\#}$, where $\underline{b}$ is the main atom of the predecessor configuration and e' is the entry in the cloud-store featuring $can_{\underline{b}}(\underline{b}, cloud(D, \Sigma, \underline{b}))$. Note that such verification is effectively possible due to condition *(2)* of Definition 7.1.
- $\mathsf{Tcheck2}$ only needs to compute the main configuration tree—the one whose configurations contain $\star$. The algorithm does *not* compute the auxiliary branches, since they are no longer necessary, as the correctness check S is done in a different way.
- The configurations of $\mathsf{Tcheck2}$ do not need to guess or memorize linear orders $\prec$ and the set S^+.

Given that $\mathsf{Tcheck2}$ is an ALOGSPACE algorithm, $\mathsf{Qcheck2}$ is an $\text{NP}^{\text{ALOGSPACE}}$ procedure. Since $\text{NP}^{\text{ALOGSPACE}} = \text{NP}^{\text{PTIME}} = \text{NP}$, query answering is in NP. In case of a fixed conjunctive query Q, since Q has a constant number of squid decompositions, $\mathsf{Qcheck2}$ runs in $\text{PTIME}^{\text{PTIME}} =$ PTIME. □

Note that the Polynomial Clouds Criterion is not syntactic. Nevertheless, it is useful for proving that query answering for some weakly guarded sets TGDs is in NP, or even in polynomial time for atomic queries. An application of this criterion is illustrated in Section 10.

The following is a direct corollary of Theorem 6.1.

Theorem 7.3. *(1) Every set Σ of GTGDs satisfies the PCC. (2) For any constant k, the class of all GTGD sets of arity bounded by k satisfies the PCC.*

The following result can be obtained by a minor adaptation of the proof of Theorem 7.2.

Theorem 7.4. *Let Σ be a fixed weakly guarded set of TGDs that enjoys the Polynomial Clouds Criterion, and let k be a constant. Then:*

(1) For a database D and a Boolean conjunctive query of treewidth $\leqslant k$, deciding whether $D \cup \Sigma \models Q$ (equivalently, $chase(D, \Sigma) \models Q$) is in PTIME.

(2) The same tractability result holds for acyclic Boolean conjunctive queries.

By analogy to the PCC, one may define various other criteria based on other bounds. In particular, we can define the *Exponential Clouds Criterion (ECC)* for classes of TGD sets, which we will use in the next section, as follows:

Definition 7.5. [Exponential Clouds Criterion] Let $\mathcal{C}$ be a class of weakly guarded TGD sets. $\mathcal{C}$ satisfies the Exponential Clouds Criterion (ECC) if both of the following conditions are satisfied:

1. There is a polynomial $\pi(\cdot)$ such that for every database D and any set of TGDs Σ in $\mathcal{C}$ of size n, $|clouds(D, \Sigma)/_{\simeq}| \leqslant 2^{\pi(|D|+n)}$.
2. There exists a polynomial $\pi'(\cdot)$ such that for every database D, any set of TGDs Σ in C of size n, and any atom $\underline{a}$:
   - if $\underline{a} \in D$, then $cloud(D, \Sigma, \underline{a})$ can be computed in time $2^{\pi'(|D|+n)}$, and
   - if $\underline{a} \notin D$, then $cloud(D, \Sigma, \underline{a})$ can be computed in time $2^{\pi'(|D|+n)}$ from D, $\underline{a}$, and $cloud(D, \Sigma, \underline{b})$, where $\underline{b}$ is the predecessor of $\underline{a}$ in $\text{gcf}(D, \Sigma)$.

We have the following result on sets of TGDs enjoying the ECC:

Theorem 7.6. *If Σ is a weakly guarded set of TGDs from a class $\mathcal{C}$ that enjoys the Exponential Clouds Criterion, then deciding for a database D and a Boolean conjunctive query Q (atomic or not) whether $D \cup \Sigma \models Q$ is in* EXPTIME.

Proof (sketch). The proof is very similar to that for Theorem 7.2. The main difference is that PTIME and ALOGSPACE are replaced by EXPTIME and APSPACE, respectively. We then get that query answering for atomic queries is in APSPACE = EXPTIME, and that answering non-atomic queries is in $\text{NP}^{\text{APSPACE}} = \text{NP}^{\text{EXPTIME}} = $ EXPTIME. Thus, in this case, there is no difference between atomic and non-atomic query answering: both are in EXPTIME. □

8. TGDs with Multiple-Atom Heads

As mentioned in Section 2, all complexity results proved so far for single-headed TGDs also carry over to the general case, where multiple atoms may appear in rule heads. We make this claim more formal here.

Theorem 8.1. *All complexity results derived in this paper for sets TGDs whose heads are single-atoms are equally valid for sets of multi-atom head TGDs.*

Proof (sketch). It suffices to show that the upper bounds carry over to the setting of TGDs with multiple-atom heads. We exhibit a transformation from an arbitrary set of TGDs Σ over a schema $\mathcal{R}$ to a set of single-headed TGDs Σ' over a schema $\mathcal{R}'$ that extends $\mathcal{R}$ with some auxiliary predicate symbols.

The TGD set Σ' is obtained from Σ by replacing each rule of the form $r : \mathit{body}(\mathbf{X}) \rightarrow \mathit{head}_1(\mathbf{Y}), \mathit{head}_2(\mathbf{Y}), \ldots, \mathit{head}_k(\mathbf{Y})$, where $k > 1$ and $\mathbf{Y}$ is the set of all the variables that appear in the head, with the following set of rules:

$$\begin{array}{rcl} \mathit{body}(\mathbf{X}) & \rightarrow & V(\mathbf{Y}) \\ V(\mathbf{Y}) & \rightarrow & \mathit{head}_1(\mathbf{Y}) \\ V(\mathbf{Y}) & \rightarrow & \mathit{head}_2(\mathbf{Y}) \\ & \vdots & \\ V(\mathbf{Y}) & \rightarrow & \mathit{head}_k(\mathbf{Y}), \end{array}$$

where V is a fresh predicate symbol, having the same arity as the number of variables in $\mathbf{Y}$. Note that, in general, neither $\mathbf{Y}$ is contained in $\mathbf{X}$ not the other way around. It is easy to see that, except for the atoms of the form $V(\mathbf{Y})$, $\mathit{chase}(D, \Sigma)$ and $\mathit{chase}(\Sigma', D)$ coincide. The atoms of the form $V(\mathbf{Y})$ have completely new predicates and thus do not match any predicate symbol in the conjunctive query Q. Therefore, $\mathit{chase}(D, \Sigma) \models Q$ iff $\mathit{chase}(\Sigma', D) \models Q$.

Obviously, Σ' can be constructed in LOGSPACE from Σ. Therefore, the extension of our complexity results to the general case is immediate, *except for the case of bounded arity.* Notice that the arity of each auxiliary predicate in the above construction depends on the number of head-variables of the corresponding transformed TGD, which, in general, is not bounded.

In case of bounded-arity WGTGDs, the EXPTIME upper bound can still be derived by the above transformation by showing that the class of TGD sets Σ' obtained by that transformation satisfies the Exponential Clouds Criterion of Section 7. To see that for each database D and each such Σ' there is only an exponential number of clouds, notice that every "large" atom $V(\mathbf{Y})$ is derived by a rule with a "small" weak guard $\underline{g}$ in its body, i.e., a weak guard $\underline{g}$ of bounded arity. The cloud $cloud(D, \Sigma', \underline{g})$ of this weak guard $\underline{g}$ clearly determines everything below $\underline{g}$ in the guarded chase forest; in particular, the cloud of $V(\mathbf{Y})$. Thus the set $clouds(D, \Sigma')$ of all clouds of all atoms is only determined by the clouds of atoms of bounded arity. For immediately verifiable combinatorial reasons, there can be only singly-exponentially many such clouds. This shows that $|clouds(D, \Sigma')/_{\simeq}|$ is singly-exponentially bounded. Therefore, the first condition of Definition 7.5 is satisfied. It is not too hard to verify the second condition of Definition 7.5, too. Thus, query-answering based on bounded-arity WGTGDs is in EXPTIME. Given that GTGDs are a subclass of WGTGDs, the same EXPTIME bound holds for bounded-arity GTGDs, as well. □

A completely different proof of the above theorem follows directly from the results by Gottlob, Manna, and Pieris (2013a) for the class of GTGDs, and from those by Gottlob, Manna, and Pieris (2013b) for the class of WGTGDs.

9. EGDs

In this section we deal with equality generating dependencies (EGDs), a generalization of *functional dependencies*, which, in turn, generalize *key dependencies* (Abiteboul, Hull, & Vianu, 1995).

Definition 9.1. Given a relational schema $\mathcal{R}$, an EGD is a first-order formula of the form $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow X_\ell = X_k$, where $\Phi(\mathbf{X})$ is a conjunction of atoms over $\mathcal{R}$, and $X_\ell, X_k \in \mathbf{X}$. Such a dependency is satisfied in an instance B if, whenever there is a homomorphism h that maps the atoms of $\Phi(\mathbf{X})$ to atoms of B, we have $h(X_\ell) = h(X_k)$.

It is possible to "repair", or chase, an instance according to EGDs by analogy with the chase based on TGDs. We start by defining the EGD chase rule.

Definition 9.2. [EGD Applicability] Consider an instance B of a schema $\mathcal{R}$, and an EGD η of the form $\Phi(\mathbf{X}) \rightarrow X_i = X_j$ over $\mathcal{R}$. We say that η is *applicable* to B if there is a homomorphism h such that $h(\Phi(\mathbf{X})) \subseteq B$ and $h(X_i) \neq h(X_j)$.

Definition 9.3. [EGD Chase Rule] Let η be an EGD of the form $\Phi(\mathbf{X}) \rightarrow X_i = X_j$ and suppose that it is applicable to an instance B via a homomorphism h. The result of the application of η on B with h is a *failure* if $\{h(X_i), h(X_j)\} \subset \Delta$ (because of the unique name assumption). Otherwise, the result of this application is the instance B' obtained from B by replacing each occurrence of $h(X_j)$ with $h(X_i)$ if $h(X_i)$ precedes $h(X_j)$ in lexicographical order. If $h(X_j)$ precedes $h(X_i)$ then the occurrences of $h(X_i)$ are replaced with $h(X_j)$ instead. We write $B \xrightarrow{\eta,h} B'$ to say that B' is obtained from B via a single EGD chase step.

Definition 9.4. [Chase sequence with respect to TGDs and EGDs] Let D be a database and $\Sigma = \Sigma_T \cup \Sigma_E$, where Σ_T is a set of TGDs and Σ_E is a set of EGDs. A (possibly infinite) *chase sequence* of D with respect to Σ is a sequence of instances $B_0, B_1, \ldots$ such that $B_i \xrightarrow{\sigma_i,h_i} B_{i+1}$, where $B_0 = D$ and $\sigma_i \in \Sigma_T \cup \Sigma_E$ for all $i \geqslant 0$. A chase sequence is said to be *failing* if its last step is a failure. A chase sequence is said to be *fair* if every TGD or EGD that is applicable at a certain step is eventually applied.

In case a fair chase sequence happens to be finite, $B_0, \ldots, B_m$, and no further rule application can change B_m, then the chase is well defined as B_m, and is denoted by $chase(D, \Sigma)$. For our purposes, the order of application of TGDs and EGDs is irrelevant. In the following therefore, when saying "the fair chase sequence", we will refer to any fair chase sequence, chosen according to some order of application of the dependencies.

It is well-known (see Johnson & Klug, 1984) that EGDs cause problems when combined with TGDs, because even for very simple types of EGDs, such as plain key constraints, the implication problem for EGDs plus TGDs and the query answering problem are undecidable. This remains true even for EGDs together with GTGDs. In fact, even though inclusion dependencies are fully guarded TGDs, the implication problem, query answering, and query containment are undecidable when keys are used as EGDs and inclusion dependencies as TGDs (Chandra & Vardi, 1985; Mitchell, 1983; Calì et al., 2003a).

Moreover, while the result of an infinite chase using TGDs is well-defined as the limit of an infinite, monotonically increasing sequence (or, equivalently, as the least fixed-point of a monotonic operator), the sequence of sets obtained in the infinite chase of a database under TGDs and EGDs is, in general, neither monotonic nor convergent. Thus, even though we can define the chase procedure for TGDs plus EGDs, it is not clear how the *result* of an infinite chase involving both TGDs and EGDs should be defined.

For the above reasons, we cannot hope to extend the positive results for weakly guarded sets of TGDs, or even GTGDs, from the previous sections to include arbitrary EGDs. Therefore, we are looking for suitable restrictions on EGDs, which would allow us to: *(i)* use the (possibly infinite) chase procedure to obtain a query-answering algorithm, and *(ii)* transfer the decidability results and upper complexity bounds derived in the previous sections to the extended formalism.

A class that fulfills both desiderata is a subclass of EGDs, which we call *innocuous* relative to a set of TGDs. These EGDs enjoy the property that query answering is insensitive to them, provided that the chase does not fail. In other words, when $\Sigma = \Sigma_T \cup \Sigma_E$, where Σ_T is a set of TGDs, Σ_E a set of EGDs, and Σ_E is innocuous relative to Σ_T, we can simply ignore these EGDs in a non-failing chase sequence. This is possible because, intuitively, such a non-failing sequence does not generate any atom that is not entailed by $chase(D, \Sigma_T)$.

More specifically, we start from the notion of *innocuous application* of an EGD. Intuitively, when making two symbols equal, an innocuous EGD application makes some atom $\underline{a}$ equal to some other existing atom $\underline{a}_0$; this way, as the only consequence of the EGD application, the original atom $\underline{a}$ is *lost*, but no new atom whatsoever is introduced. The concept of innocuous EGD application is formally defined as follows.

Definition 9.5. [Innocuous EGD application] Consider a (possibly infinite) non-failing chase sequence $D = B_0, B_1, \ldots$, starting with a database D, with respect to a set $\Sigma = \Sigma_T \cup \Sigma_E$, where Σ_T is a set of TGDs and Σ_E is a set of EGDs. We say that the EGD application $B_i \xrightarrow{\eta,h} B_{i+1}$, where $\eta \in \Sigma_E$ and $i \geqslant 0$, is *innocuous* if $B_{i+1} \subset B_i$.

Notice that innocuousness is a *semantic*, not syntactic, property. It is desirable to have innocuous EGD applications because such applications cannot trigger new TGD applications, i.e., TGD applications that were not possible before the EGD was applied.

Given that it might be undecidable whether a set of dependencies from a certain class guarantees innocuousness of all EGD applications, one can either give a direct proof of innocuousness for a concrete set of dependencies, as we will do in Section 10.2, or define sufficient syntactic conditions that guarantee innocuousness of EGD applications for an entire class of dependencies, as done, e.g., by Calì et al. (2012a).

Definition 9.6. Let $\Sigma = \Sigma_T \cup \Sigma_E$, where Σ_T is a set of TGDs and Σ_E a set of EGDs, where $\Sigma = \Sigma_T \cup \Sigma_E$. Σ_E is *innocuous* for Σ_T if, for every database D such that the fair chase sequence of D with respect to Σ is non-failing, each application of an EGD in such sequence of D with respect to Σ is innocuous.

Theorem 9.7. *Let $\Sigma = \Sigma_T \cup \Sigma_E$, where Σ_T is a set of TGDs and Σ_E a set of EGDs that is innocuous for Σ_T. Let D be a database such that the fair chase sequence of D with respect to Σ is non-failing. Then $D \cup \Sigma \models Q$ iff $chase(D, \Sigma_T) \models Q$.*

Proof. Consider the fair chase sequence $B_0, B_1, \ldots$ of $D = B_0$ in the presence of Σ, where $B_i \xrightarrow{\sigma_i, h_i} B_{i+1}$ for $i \geqslant 0$ and $\sigma \in \Sigma_T \cup \Sigma_E$. Let us define a modified chase procedure which we call the *blocking chase*, denoted by $blockchase(D, \Sigma)$. The blocking chase uses two sets: a set C of blocked atoms and a set of (unblocked) atoms A. When started on a database D such that $D \models \Sigma_E$ (the case $D \not\models \Sigma_E$ is not possible as this implies an immediate chase failure), C is initialized to the empty set ($C = \varnothing$) and A is initialized to D. After the initialization, the blocking chase attempts to apply the dependencies in $\Sigma_T \cup \Sigma_E$ exactly in the same way as in the standard fair chase sequence, with the following caveats. While trying an application of $\langle \sigma_i, h_i \rangle$:

- If σ_i is a TGD, and if $h_i(body(\sigma_i)) \cap C = \varnothing$, then apply $\langle \sigma_i, h_i \rangle$ and add the new atom generated by this application to A.
- If σ_i is a TGD and $h_i(body(\sigma_i)) \cap C \neq \varnothing$, then the application of $\langle \sigma_i, h_i \rangle$ is blocked, and nothing is done.
- If σ_i is an EGD, then the application of $\langle \sigma_i, h_i \rangle$ proceeds as follows. Add to C all the facts that in the standard chase disappear in that step (because $B_i \subseteq B_{i-1}$, due to the innocuousness), i.e., add to C the set $B_i - B_{i-1}$. Thus, instead of eliminating tuples from A, the blocking chase simply bans them from being used by putting them in C.

Note that, by the construction of $blockchase(D, \Sigma)$, whenever the block chase encounters an EGD σ_i, $\langle \sigma_i, h_i \rangle$ is actually applicable, so $blockchase(D, \Sigma)$ is well-defined. Let us use C_i and A_i to denote the values of C and A at step i, respectively. Initially, $C_0 = \varnothing$ and $A_0 = D$ as explained before. Observe that $\varnothing = C_0 \subseteq C_1 \subseteq C_2 \subseteq \cdots$ and $D = A_0 \subseteq A_1 \subseteq A_2 \subseteq \cdots$ are monotonically increasing sequences that have least upper bounds $C^* = \cup_i C_i$ and $A^* = \cup_i A_i$, respectively. Clearly, (C^*, A^*) is the least fixpoint of the transformation performed by $blockchase(D, \Sigma)$ (with respect to component-wise set inclusion).

Now, let S be defined as $S = A^* - C^*$. By the definition of S, we have: $S \models \Sigma$. Moreover, there is a homomorphism h that maps $chase(D, \Sigma_T)$ to S. Note that h is the limit homomorphism of the sequence $h_1, h_2, h_3, \ldots$ (these h_is are the very homomorphisms used while computing the block chase), and can be defined as the set of all pairs (x, y) such that there exists an $i \geqslant 0$ such that $h_i(h_{i-1}(\cdots h_1(x))) = y$ and y is not altered by any homomorphism h_j for $j > i$. Note that for every instance B that contains D, we have $B \models D$. In particular, $S \models D$. Putting everything together, we conclude that $S \models D \cup \Sigma$.

It is also well-known (see Nash et al., 2006) that for any set of atoms M such that $M \models S \cup \Sigma_T$, there is a homomorphism h_M such that $h_M(\mathit{chase}(D, \Sigma_T)) \subseteq M$. Now assume $D \cup \Sigma \models Q$. Then $S \models Q$ and, because $S \subseteq \mathit{chase}(D, \Sigma_T)$, we also have that $\mathit{chase}(D, \Sigma_T) \models Q$. Conversely, if $\mathit{chase}(D, \Sigma_T) \models Q$, then there is a homomorphism g, such that $g(Q) \subseteq \mathit{chase}(D, \Sigma_T)$. Therefore, for any set of atoms M such that $M \models D \cup \Sigma$, since $h_M(\mathit{chase}(D, \Sigma_T)) \subseteq M$, we have $h_M(g(Q)) \subseteq M$. The latter means that $M \models Q$. $\square$

We now come to the problem of checking, given a database D and a set $\Sigma = \Sigma_T \cup \Sigma_E$, where Σ_T is set of WGTGDs and Σ_E are EGDs innocuous for Σ_T, whether the fair chase sequence (denoted $B_0, B_1, \ldots$) of D with respect to Σ fails. Consider an application $B_i \xrightarrow{\eta,h} B_{i+1}$, with $\eta \in \Sigma_E$ of the form $\Phi(\mathbf{X}) \to X_\ell = X_k$. When this application causes the chase to fail, we have that $h(X_\ell)$ and $h(X_k)$ are distinct values in $\mathit{dom}(D)$. Notice that B_j exists for $j \leqslant i$, while it does not exist for any $j > i$.

Lemma 9.8. *Consider a database D and a set of dependencies $\Sigma = \Sigma_T \cup \Sigma_E$, where Σ_T is a weakly guarded set of TGDs and Σ_E are EGDs that are innocuous for Σ_T. Then the fair chase sequence of D with respect to Σ fails iff there is an EGD $\eta \in \Sigma_E$ of the form $\Phi(\mathbf{X}) \to X_\ell = X_k$ and a homomorphism h such that $h(\Phi(\mathbf{X})) \subseteq \mathit{chase}(D, \Sigma_T)$, $h(X_\ell) \neq h(X_k)$, and $\{h(X_\ell), h(X_k)\} \subseteq \mathit{dom}(D)$.*

Proof (sketch).

"If". Let $B_0, B_1, \ldots$ be the fair chase sequence of D with respect to Σ. First, it is not difficult to show that, since Σ_E is innocuous relative to Σ_T, if the failure occurs at step ℓ then all EGD applications $B_i \xrightarrow{\sigma_i,h_i} B_{i+1}$, such that $\sigma_i \in \Sigma_E$ and $i < \ell - 1$, are innocuous (see a similar proof by Calì, Console, & Frosini, 2013) in the sequence $B_0, \ldots, B_{\ell-1}$. From this, the "if" direction follows straightforwardly.

"Only if". By assumption, η fails at some B_k, $k \geqslant 1$. Since applications of innocuous EGDs can only remove tuples from the chase, it is easily seen that, if η is applicable to B_k via an homomorphism h, then it is also applicable to $\mathit{chase}(D, \Sigma_T)$ via the same homomorphism h, which settles the "only-if" part. $\square$

Theorem 9.9. *Consider a database D and a set of dependencies $\Sigma = \Sigma_T \cup \Sigma_E$, where Σ_T are GTGDs (resp., WGTGDs) and Σ_E are EGDs that are innocuous for Σ_T. Checking whether the fair chase sequence of D with respect to Σ fails is decidable, and has the same complexity as query answering for GTGDs (resp., WGTGDs) alone.*

Proof (sketch). Let neq be a new binary predicate, which will serve as inequality. The extension of neq is defined as $\mathit{dom}(D) \times \mathit{dom}(D) - \{(d, d) \mid d \in \mathit{dom}(D)\}$ and can be constructed in time quadratic in $|\mathit{dom}(D)|$. Now, for every EGD η of the form $\Phi(\mathbf{X}) \to X_1 = X_2$, where $X_1, X_2 \in \mathbf{X}$, we define the following Boolean conjunctive query (expressed as a set of atoms): $Q_\eta = \Phi(\mathbf{X}) \cup \{\mathit{neq}(X_1, X_2)\}$. Since, by construction, no new facts of the form $\mathit{neq}(\sigma_1, \sigma_2)$ are introduced in the chase, it is immediate to see, from Lemma 9.8, that at least one of the above Q_η has a positive answer if and only if the fair chase sequence of D with respect to Σ fails. By Theorem 9.7, answering the query Q_η can be done with respect to the chase by Σ_T alone, which is decidable. $\square$

Let $\Sigma = \Sigma_T \cup \Sigma_E$ be as in the above theorem, D be a database, and let Q be a query. By the above theorem, we can check $\Sigma \cup D \models Q$ with the help of the following algorithm:

1. check whether the fair chase sequence of D with respect to Σ fails with the algorithm described in Theorem 9.9;
2. if the fair chase sequence of D with respect to Σ fails, then return *"true"* and halt;
3. if $D \cup \Sigma_T \models Q$ then return *"true"*; otherwise return *"false"*.

This gives us the following corollary:

Corollary 9.10. *Answering general conjunctive queries under weakly guarded sets of TGDs and innocuous EGDs is* PTIME *reducible to answering queries of the same class under a weakly guarded sets of TGDs alone, and thus has the same complexity.*

10. Applications

In this section we discuss applications of our results on weakly guarded sets of TGDs to Description Logic languages and object-oriented logic languages.

10.1 DL-Lite

DL-Lite (Calvanese et al., 2007; Artale et al., 2009) is a prominent family of ontology languages that has tractable query answering. Interestingly, a restriction of GTGDs called *linear TGDs* (which have exactly one body-atom and one head-atom) properly extends most DL-Lite languages, as shown by Calì et al. (2012a). The complexity of query answering under linear TGDs is lower than that of GTGDs, and we refer the reader to the work of Calì et al. (2012a) for more details.

Furthermore, Calì et al. (2012a) also show that the language of GTGDs properly extends the description logic $\mathcal{EL}$ as well as its extension $\mathcal{ELI}^f$, which allows inverse and functional roles. The fact that TGDs capture important DL-based ontology languages confirms that TGDs are useful tools for ontology modeling and querying.

10.2 F-Logic Lite

F-Logic Lite is an expressive subset of F-logic (Kifer et al., 1995), a well-known formalism introduced for object-oriented deductive languages. We refer the reader to the work by Calì and Kifer (2006) for details about F-Logic Lite. Roughly speaking, compared to full F-Logic, F-Logic Lite excludes negation and default inheritance, and allows only a limited form of cardinality constraints. F-Logic Lite can be encoded by a set of twelve TGDs and EGDs, below, which we denote by Σ_{FLL}:

ρ_1: $\mathsf{type}(O, A, T), \mathsf{data}(O, A, V) \rightarrow \mathsf{member}(V, T)$.
ρ_2: $\mathsf{sub}(C_1, C_3), \mathsf{sub}(C_3, C_2) \rightarrow \mathsf{sub}(C_1, C_2)$.
ρ_3: $\mathsf{member}(O, C), \mathsf{sub}(C, C_1) \rightarrow \mathsf{member}(O, C_1)$.

ρ_4: $\mathsf{data}(O, A, V), \mathsf{data}(O, A, W), \mathsf{funct}(A, O) \rightarrow V = W$.
Note that this is the only EGD in this axiomatization.
ρ_5: $\mathsf{mandatory}(A, O) \rightarrow \exists V\, \mathsf{data}(O, A, V)$.
Note that this TGD has an existentially quantified variable in the head.
ρ_6: $\mathsf{member}(O, C), \mathsf{type}(C, A, T) \rightarrow \mathsf{type}(O, A, T)$.
ρ_7: $\mathsf{sub}(C, C_1), \mathsf{type}(C_1, A, T) \rightarrow \mathsf{type}(C, A, T)$.
ρ_8: $\mathsf{type}(C, A, T_1), \mathsf{sub}(T_1, T) \rightarrow \mathsf{type}(C, A, T)$.
ρ_9: $\mathsf{sub}(C, C_1), \mathsf{mandatory}(A, C_1) \rightarrow \mathsf{mandatory}(A, C)$.
ρ_{10}: $\mathsf{member}(O, C), \mathsf{mandatory}(A, C) \rightarrow \mathsf{mandatory}(A, O)$.
ρ_{11}: $\mathsf{sub}(C, C_1), \mathsf{funct}(A, C_1) \rightarrow \mathsf{funct}(A, C)$.
ρ_{12}: $\mathsf{member}(O, C), \mathsf{funct}(A, C) \rightarrow \mathsf{funct}(A, O)$.

The results of this paper apply to the above set of constraints, since Σ_{FLL} is a weakly guarded set, and the single EGD ρ_4 is innocuous. The innocuousness of ρ_4 is shown by observing that, whenever the EGD is applied, it turns one atom into another; moreover, all new data atoms created in the chase (see rule ρ_5) have new labeled nulls exactly in the position $\mathsf{data}[3]$, where the symbols to be equated also reside.

We now prove the relevant complexity results. We start by showing that BCQ answering under F-Logic Lite is NP-complete.

Theorem 10.1. *Conjunctive query answering under F-Logic Lite rules is* NP*-hard.*

Proof (sketch). The proof is by reduction from the 3-COLORABILITY problem. Encode a graph $G = (V, E)$ as a conjunctive query Q which, for each edge (v_i, v_j) in E, has two atoms $\mathsf{data}(X, V_i, V_j)$ and $\mathsf{data}(X, V_j, V_i)$, where X is a unique variable. Let D be the database $D = \{\mathsf{data}(o, r, g),\ \mathsf{data}(o, g, r),\ \mathsf{data}(o, r, b),\ \mathsf{data}(o, b, r),\ \mathsf{data}(o, g, b),\ \mathsf{data}(o, b, g)\}$. Then, G is three-colorable iff $D \models Q$, which is the case iff $D \cup \Sigma_{FLL} \models Q$. The transformation from G to (Q, D) is obviously polynomial, which proves the claim. $\square$

Theorem 10.2. *Conjunctive query answering under F-Logic Lite rules is in* NP.

Proof (sketch). As mentioned before, we can ignore the only EGD in Σ_{FLL} since, being innocuous, it does not interfere with query answering. Let Σ'_{FLL} denote the set of TGDs resulting from Σ_{FLL} by eliminating rule ρ_4, i.e., let $\Sigma'_{FLL} = \Sigma_{FLL} - \{\rho_4\}$. To establish membership in NP, it is sufficient to show that: *(1)* Σ'_{FLL} is weakly guarded; *(2)* Σ'_{FLL} enjoys the PCC (see Definition 7.1). Under the above condition, the membership in NP can be proved by exhibiting the following. *(i)* An algorithm, analogous to Acheck, that constructs *all* "canonical" versions of the atoms of the chase and their clouds (which are stored in a "cloud store"), in polynomial time. Then the algorithm should check whether an atomic (Boolean) query is satisfied by an atom in the cloud store. *(ii)* An algorithm, analogous to Qcheck, that guesses (by calling an analogous version of Tcheck) entire clouds through guessing the cloud index (a unique integer) in the cloud store. Then the algorithm should check, in alternating logarithmic space (ALOGSPACE), the correctness of the cloud guess. In that check, it can use only the cloud of the main atom of the predecessor configuration. The complexity of running this algorithm is shown to be $\text{NP}^{\text{ALOGSPACE}} = \text{NP}$.

(1) is easy: the affected positions are $\mathsf{data}[3]$, $\mathsf{member}[1]$, $\mathsf{type}[1]$, $\mathsf{mandatory}[2]$, $\mathsf{funct}[2]$ and $\mathsf{data}[1]$. It is easy to see that every rule of Σ'_{FLL} is weakly guarded, and thus Σ_{FLL} is weakly guarded.

Now let us sketch *(2)*. We need to show that Σ'_{FLL} satisfies the two conditions of Definition 7.1. We prove that the first condition holds for Σ'_{FLL} as follows. Let $\Sigma^{full}_{FLL} = \Sigma'_{FLL} - \{\rho_5\}$. These are all full TGDs (no existentially-quantified variables) and their application does not alter the domain. We have $chase(D, \Sigma'_{FLL}) = chase(chase(D, \Sigma^{full}_{FLL}), \Sigma'_{FLL})$. Let us now have a closer look at $D^+ = chase(D, \Sigma^{full}_{FLL})$. Clearly, $dom(D^+) = dom(D)$. For each predicate symbol p, let $Rel(p)$ denote the relation consisting of all the p-atoms in D^+. Let Ω be the family of all the relations that can be obtained from any of the relations $Rel(p)$ by performing an arbitrary selection followed by some projection (we forbid disjunctions in the selection predicate). For example, let $c, d \in dom(D)$. Then Ω will contain the relations $\pi_{1,2}(\sigma_{\{1=c\}} Rel(\mathsf{data}))$, $\pi_2(\sigma_{\{1=d \wedge 3=c\}} Rel(\mathsf{data}))$, and so on, where the numbers represent the attributes to which selection is applied. Given that D^+ is of size polynomial in D and that the maximum arity of any relation $Rel(p)$ is 3, the set Ω is of size polynomial in D^+ and thus polynomial in D. It can now be shown that Ω is preserved in a precise sense, when going to the final result $chase(D^+, \Sigma'_{FLL})$: for each relation $Rel'(p)$ corresponding to predicate p in the final chase result, when performing a selection on values outside of $dom(D)$ and projecting on the columns not used in the selection, the set of all tuples of $dom(D)$-elements in the result is a relation in Ω. For example, if v_5 is a labeled null, then the set of all $T \in dom(D)$, such that $\mathsf{member}(v_5, T)$ is an element of the final result, is a relation in Ω. Similarly, if v_7 and v_8 are new values, the set of all values A, such that $\mathsf{data}(v_7, A, v_8)$ is in the chase, is a relation in Ω. From this it follows that Σ'_{FLL} satisfies *(2)*. In fact, all possible clouds are determined by the polynomially many ways of choosing at most three elements of Ω for each predicate. The proof of the preservation property can be done by induction on the i-th new labeled null added. Roughly, for each such labeled null, created by rule ρ_5, we just analyze which sets of values (or tuples) are attached to it via rules ρ_4, then ρ_6, ρ_7, ρ_8, ρ_{10}, and so on, and conclude that these sets were already present at the next lower level, and thus, by induction hypothesis, are in Ω.

The second condition of Definition 7.1 is proved by similar arguments. $\square$

From Theorems 10.1 and 10.2 we immediately get the following result.

Corollary 10.3. *Conjunctive query answering under F-Logic Lite rules is* NP*-complete for general conjunctive queries, and in* PTIME *for fixed-size or atomic conjunctive queries.*

11. Conclusions and Related Work

In this paper we identified a large and non-trivial class of *tuple-generating* and *equality-generating* dependencies for which the problems of conjunctive query containment and answering are decidable, and provided the relevant complexity results. Applications of our results span databases and knowledge representation. In particular, we have shown that this class of constraints subsumes the classical work by Johnson and Klug (1984) as well as more recent results from Calì and Kifer (2006). Moreover, we are able to capture relevant ontology formalisms in the Description Logics (DL) family, in particular DL-Lite and $\mathcal{EL}$.

The problem of query containment for non-terminating chase was addressed in the database context by Johnson and Klug (1984), where the ontological theory contains inclusion dependencies and key dependencies of a particular form. The introduction of the *DL-Lite* family of description logics in the works of Calvanese et al. (2007) and Artale et al.

(2009) was a significant leap forward in ontological query answering due to the expressiveness of DL-Lite languages and their tractable data complexity. Conjunctive query answering in DL-Lite has the advantage of being *first-order rewritable*, i.e., any pair $\langle Q, \Sigma \rangle$, where Q is a CQ and Σ is a DL-Lite ontology (TBox), can be rewritten as a first-order query Q_Σ such that, for every database (ABox) D, the answer to Q against the logical theory $D \cup \Sigma$ coincides with the answer to Q_Σ against D. Since each first-order query can be written in SQL, in practical terms this means that a pair $\langle Q, \Sigma \rangle$ can be rewritten as an SQL query over the original database D.

Rewritability is widely adopted in ontology querying. The works by Calì, Calvanese, De Giacomo, and Lenzerini (2001), and by Calì, Lembo, and Rosati (2003b) present query rewriting techniques that deal with Entity-Relationship schemata and inclusion dependencies, respectively. The work by Pérez-Urbina, Motik, and Horrocks (2010) presents a Datalog rewriting algorithm for the expressive DL $\mathcal{ELHIO}^\neg$, which comprises a limited form of concept and role negation, role inclusion, inverse roles, and nominals, i.e., concepts that are interpreted as singletons. Conjunctive query answering in $\mathcal{ELHIO}^\neg$ is PTIME-complete in data complexity, and the proposed algorithm is also optimal for other ontology languages such as DL-Lite. Optimizations of rewriting under linear TGDs (TGDs with exactly one atom in the body) are presented by Gottlob, Orsi, and Pieris (2011), and by Orsi and Pieris (2011). Gottlob and Schwentick (2012) showed that the rewriting of a conjunctive query under a set of linear TGDs can be of polynomial size in the query and the TGD set.

Other rewriting techniques for PTIME-complete languages (in data complexity) have been proposed for the description logic $\mathcal{EL}$ (Rosati, 2007; Lutz, Toman, & Wolter, 2009; Krötzsch & Rudolph, 2007). Another approach worth mentioning is a combination of rewriting and of the chase (see Kontchakov, Lutz, Toman, Wolter, & Zakharyaschev, 2010); this technique was introduced for DL-Lite in order to tackle the performance problems that arise when the rewriting according to the ontology is too large.

Recent works concentrate on semantic characterization of sets of TGDs under which query answering is decidable (Baget et al., 2011a). The notion of first-order rewritability is tightly connected to that of *finite unification set* (FUS). A FUS is semantically characterized as a set of TGDs that enjoy the following property: for every conjunctive query Q, the rewriting Q_Σ of Q obtained by backward-chaining through unification, according to the rules in Σ, terminates. Another semantic characterization of TGDs is that of *bounded treewidth set* (BTS), i.e., a set of TGDs such that the chase under such TGDs has bounded treewidth. As seen in Section 3, every weakly guarded set of TGDs is a BTS. A *finite expansion set* (FES) is a set of TGDs that guarantees, for every database, the termination of the *restricted* chase, and therefore the decidability of query answering.

The Datalog$^\pm$ family (Calì et al., 2011) has been proposed with the purpose of providing tractable query answering algorithms for more general ontology languages. In Datalog$^\pm$, the fundamental constraints are TGDs and EGDs. Clearly, TGDs are an extension of Datalog rules. The absence of value invention (existential quantification in the head), thoroughly discussed by Patel-Schneider and Horrocks (2007), is the main shortcoming of plain Datalog in modeling ontologies and even conceptual data formalisms such as the Entity-Relationship model (Chen, 1976). Sets of GTGDs or WGTGDs are Datalog$^\pm$ ontologies. Datalog$^\pm$ languages easily extend the most common tractable ontology languages; in particular, the main DL-Lite languages (see Calì et al., 2012a). The fundamental decidability paradigms in the Datalog$^{\pm}$ family are the following:

- *Chase termination.* When the chase terminates, a finite instance is produced; obviously, by Theorem 2.10, query answering in such a case is decidable. The most notable syntactic restriction guaranteeing chase termination is *weak acyclicity* of TGDs, for which we refer the reader to the milestone paper of Fagin et al. (2005). More general syntactic restrictions are studied by Deutsch, Nash, and Remmel (2008), Marnette (2009), Greco, Spezzano, and Trubitsyna (2011), Baget et al. (2011a), and Grau, Horrocks, Krötzsch, Kupke, Magka, Motik, and Wang (2012). A semantic property of TGDs, called *parsimony*, is introduced by Leone, Manna, Terracina, and Veltri (2012). Parsimony ensures decidability of query answering by termination of a special version of chase, called *parsimonious* chase.
- *Guardedness.* This is the paradigm we studied in this paper. A thorough study of the data complexity of query answering under GTGDs and *linear TGDs*, a subset of the guarded class, is found in the work by Calì et al. (2012a). The interesting classes of *frontier guarded (FGTGDs)* and *weakly frontier-guarded TGDs (WFGTGDs)* were considered and studied by Baget et al. (2011a), Baget, Mugnier, Rudolph, and Thomazo (2011b), and Krötzsch and Rudolph (2011). The idea underlying these classes is that, to obtain decidability, it is sufficient to guard only *frontier variables*, that is, variables that occur both in the body and in the head of a rule.[8] WFGTGDs are syntactically more liberal and more succinct than WGTGDs, but conjunctive query answering under WFGTGDs is computationally more expensive in case of bounded arities. It can be seen that querying under WFGTGDs is no more expressive than querying under WGTGDs. In fact, for every WFGTGD set Σ and CQ Q, there exists a WGTGD set Σ' and a CQ Q' such that for every database D, $D \cup \Sigma \models Q$ iff $D \cup \Sigma' \models Q'$. A generalization of WFGTGDs, called *greedy bounded-treewidth TGDs*, was proposed by Baget et al. (2011b), together with a complexity analysis. The guardedness paradigm has been combined with acyclicity by Krötzsch and Rudolph (2011), where a generalization of both WFGTGDs and weakly acyclic TGDs is proposed.
- *Stickiness.* The class of *sticky sets of TGDs* (or *sticky Datalog*$^{\pm}$, see Calì et al., 2012b) is defined by means of syntactic restriction on the rule bodies, which ensure that each sticky set of TGDs is first-order rewritable, being a FUS, according to Baget et al. (2011a). Civili and Rosati (2012) have proposed an extension of sticky sets of TGDs.

The interaction between equality generating dependencies and TGDs has been the subject of several works, starting from the work of Johnson and Klug (1984), which deals with functional and inclusion dependencies, proposing a class of inclusion dependencies called *key-based*, which, intuitively, has no interaction with key dependencies thanks to syntactic restrictions. The absence of interaction between EGDs and TGDs is captured by the notion of *separability*, first introduced by Calì et al. (2003a) for key and inclusion dependencies, and also adopted, though sometimes not explicitly stated, for instance, by Calì, Gottlob, and Pieris (2012a), Artale et al. (2009) and Calvanese et al. (2007)—see the work by Calì, Gottlob, Orsi, and Pieris (2012b) for a survey on the topic.

8. FGTGDs were independently discovered by Mantas Šimkus while working on his doctoral thesis.

As shown by Calì et al. (2012a), stratified negation can be added straightforwardly to Datalog$^{\pm}$. More recently, guarded Datalog$^{\pm}$ was extended by two versions of well-founded negation (see Gottlob, Hernich, Kupke, & Lukasiewicz, 2012; Hernich, Kupke, Lukasiewicz, & Gottlob, 2013).

In ontological query answering, normally both finite and infinite models of theories are considered. In some cases, restricting the attention to finite solutions (models) only is not always equivalent to the general approach. The property of equivalence between query answering under finite models and query answering under arbitrary models (finite and infinite) is called *finite controllability*, and it was proved for restricted classes of functional and inclusion dependencies by Johnson and Klug (1984). Finite controllability was proved for the class of arbitrary inclusion dependencies in a pioneering work by Rosati (2011). An even more general result appears in the work of Bárány et al. (2010), where it is shown that finite controllability holds for guarded theories.

A related previous approach to guarded logic programming is *guarded open answer set programming* (Heymans, Nieuwenborgh, & Vermeir, 2005). It is easy to see that a set of GTGDs can be interpreted as a guarded answer set program, as defined by Heymans et al. (2005), but guarded answer set programs are more expressive than GTGDs because they allow negation.

Implementations of ontology-based data access systems take advantage of query answering techniques for tractable ontologies; in particular, we mention DLV$^{\exists}$ (Leone et al., 2012), Mastro (Savo, Lembo, Lenzerini, Poggi, Rodriguez-Muro, Romagnoli, Ruzzi, & Stella, 2010) and NYAYA (De Virgilio, Orsi, Tanca, & Torlone, 2012).

Acknowledgments

This is the extended version of results by the same authors, published in the KR 2008 Conference and in the DL 2008 Workshop. Andrea Calì and Georg Gottlob are also affiliated with the Oxford-Man Institute of Quantitative Finance, University of Oxford, UK. Andrea Calì acknowledges support by the EPSRC project "Logic-based Integration and Querying of Unindexed Data" (EP/E010865/1). Georg Gottlob acknowledges funding from the European Research Council under the European Community's Seventh Framework Program (FP7/2007-2013) / ERC grant agreement DIADEM no. 246858. Michael Kifer was partially supported by the NSF grant 0964196. The authors are grateful to Andreas Pieris, Marco Manna, Michael Morak and the anonymous reviewers for their valuable comments and suggestions to improve the paper.

References

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

Adler, I., Gottlob, G., & Grohe, M. (2007). Hypertree width and related hypergraph invariants. *Eur. Journal of Combinatorics*, *28*(8), 2167–2181.

Aho, A., Sagiv, Y., & Ullman, J. D. (1979). Equivalence of relational expressions. *SIAM Journal of Computing*, *8*(2), 218–246.

Arenas, M., Bertossi, L. E., & Chomicki, J. (1999). Consistent query answers in inconsistent databases. In *Proc of PODS 1999*, pp. 68–79.

Artale, A., Calvanese, D., Kontchakov, R., & Zakharyaschev, M. (2009). The DL-lite family and relations. *J. Artif. Intell. Res.*, *36*, 1–69.

Baader, F., Brandt, S., & Lutz, C. (2005). Pushing the $\mathcal{EL}$ envelope. In *Proc. of IJCAI 2005*, pp. 364–369.

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

Baget, J.-F., Mugnier, M.-L., Rudolph, S., & Thomazo, M. (2011b). Walking the complexity lines for generalized guarded existential rules. In *Proc. of IJCAI 2011*, pp. 712–717.

Bárány, V., Gottlob, G., & Otto, M. (2010). Querying the guarded fragment. In *Proc. of LICS 2010*, pp. 1–10.

Beeri, C., Fagin, R., Maier, D., Mendelzon, A. O., Ullman, J. D., & Yannakakis, M. (1981). Properties of acyclic database schemes. In *Proc. of STOC 1981*, pp. 355–362.

Beeri, C., & Vardi, M. Y. (1981). The implication problem for data dependencies. In *Proc. of ICALP 1981*, pp. 73–85.

Bourhis, P., Morak, M., & Pieris, A. (2013). The impact of disjunction on query answering under guarded-based existential rules. In *Proc. of IJCAI 2013*.

Cabibbo, L. (1998). The expressive power of stratified logic programs with value invention. *Inf. Comput.*, *147*(1), 22–56.

Calì, A., Calvanese, D., De Giacomo, G., & Lenzerini, M. (2001). Accessing data integration systems through conceptual schemas. In *Proc. of ER 2001*, pp. 270–284.

Calì, A., Console, M., & Frosini, R. (2013). On separability of ontological constraints. Forthcoming.

Calì, A., Gottlob, G., & Kifer, M. (2008). Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of KR 2008*, pp. 70–80.

Calì, A., Gottlob, G., & Lukasiewicz, T. (2009). A general datalog-based framework for tractable query answering over ontologies. In *Proc. of PODS 2009*, pp. 77–86.

Calì, A., Gottlob, G., & Lukasiewicz, T. (2012a). A general datalog-based framework for tractable query answering over ontologies. *J. Web Semantics*, *14*, 57–83. Extended version of (Calì, Gottlob, & Lukasiewicz, 2009).

Calì, A., Gottlob, G., Orsi, G., & Pieris, A. (2012b). On the interaction of existential rules and equality constraints in ontology querying. In *Proc. of Correct Reasoning 2012*, pp. 117–133.

Calì, A., Gottlob, G., & Pieris, A. (2011). New expressive languages for ontological query answering. In *Proc. of AAAI 2011*.

Calì, A., Gottlob, G., & Pieris, A. (2012a). Ontological query answering under expressive entity-relationship schemata. *Inf. Syst.*, *37*(4), 320–335.

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

Calì, A., & Kifer, M. (2006). Containment of conjunctive object meta-queries. In *Proc. of VLDB 2006*, pp. 942–952.

Calì, A., Lembo, D., & Rosati, R. (2003a). On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS 2003*, pp. 260–271.

Calì, A., Lembo, D., & Rosati, R. (2003b). Query rewriting and answering under constraints in data integration systems. In *Proc. of IJCAI 2003*, pp. 16–21.

Calì, A., & Martinenghi, D. (2010). Querying incomplete data over extended er schemata. *TPLP*, *10*(3), 291–329.

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

Calvanese, D., De Giacomo, G., & Lenzerini, M. (2002). Description logics for information integration. In *Computational Logic: Logic Programming and Beyond*, Vol. 2408 of *LNCS*, pp. 41–60. Springer.

Calvanese, D., De Giacomo, G., & Lenzerini, M. (1998). On the decidability of query containment under constraints. In *Proc. of PODS 1998*, pp. 149–158.

Chandra, A. K., Kozen, D., & Stockmeyer, L. J. (1981a). Alternation. *J. of the ACM*, *28*(1), 114–133.

Chandra, A. K., Lewis, H. R., & Makowsky, J. A. (1981b). Embedded implicational dependencies and their inference problem. In *Proc. of STOC 1981*, pp. 342–354.

Chandra, A. K., & Merlin, P. M. (1977). Optimal implementation of conjunctive queries in relational data bases. In *Proc. of STOC 1977*, pp. 77–90.

Chandra, A. K., & Vardi, M. Y. (1985). The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. Comput.*, *14*, 671–677.

Chen, P. P. (1976). The entity-relationship model - toward a unified view of data. *Trans. Database Syst.*, *1*(1), 9–36.

Civili, C., & Rosati, R. (2012). A broad class of first-order rewritable tuple-generating dependencies. In *Proc. of Datalog 2.0 2012*, pp. 68–80.

Courcelle, B. (1990). The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, *85*(1), 12–75.

Dantsin, E., Eiter, T., Gottlob, G., & Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Computing Surveys*, *33*(3), 374–425.

De Virgilio, R., Orsi, G., Tanca, L., & Torlone, R. (2012). NYAYA: A system supporting the uniform management of large sets of semantic data. In *Proc. of ICDE 2012*, pp. 1309–1312.

Deutsch, A., Nash, A., & Remmel, J. B. (2008). The chase revisited. In *Proc. of PODS 2008*, pp. 149–158.

Fagin, R. (1983). Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, *30*(3), 514–550.

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

Goncalves, M. E., & Grädel, E. (2000). Decidability issues for action guarded logics. In *Proc. of DL 2000*, pp. 123–132.

Gottlob, G., Hernich, A., Kupke, C., & Lukasiewicz, T. (2012). Equality-friendly well-founded semantics and applications to description logics. In *Proc. of AAAI 2012*.

Gottlob, G., Leone, N., & Scarcello, F. (2001). Hypertree decompositions: A survey. In *Proc. of MFCS 2001*, pp. 37–57.

Gottlob, G., Leone, N., & Scarcello, F. (2002). Hypertree decompositions and tractable queries. *J. Comp. Syst. Sci.*, *64*(3).

Gottlob, G., Leone, N., & Scarcello, F. (2003). Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. Syst. Sci.*, *66*(4), 775–808.

Gottlob, G., Manna, M., & Pieris, A. (2013a). Combining decidability paradigms for existential rules. To appear in TPLP.

Gottlob, G., Manna, M., & Pieris, A. (2013b). Querying hybrid fragments of existential rules. Forthcoming.

Gottlob, G., & Nash, A. (2006). Data exchange: computing cores in polynomial time. In *Proc. of PODS 2006*, pp. 40–49.

Gottlob, G., Orsi, G., & Pieris, A. (2011). Ontological queries: Rewriting and optimization. In *Proc. of ICDE 2011*, pp. 2–13.

Gottlob, G., & Schwentick, T. (2012). Rewriting ontological queries into small nonrecursive datalog programs. In *Proc. of KR 2012*.

Grädel, E. (1999). On the restraining power of guards. *J. Symb. Log.*, *64*(4), 1719–1742.

Grau, B. C., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., & Wang, Z. (2012). Acyclicity conditions and their application to query answering in description logics. In *Proc. of KR 2012*.

Greco, S., Spezzano, F., & Trubitsyna, I. (2011). Stratification criteria and rewriting techniques for checking chase termination. *PVLDB*, *4*(11), 1158–1168.

Hernich, A., Kupke, C., Lukasiewicz, T., & Gottlob, G. (2013). Well-founded semantics for extended datalog and ontological reasoning. In *Proc. of PODS 2013*, pp. 225–236.

Hernich, A., Libkin, L., & Schweikardt, N. (2011). Closed world data exchange. *ACM Trans. Database Syst.*, *36*(2), 14–53.

Heymans, S., Nieuwenborgh, D. V., & Vermeir, D. (2005). Guarded open answer set programming. In *Proc. of LPNMR 2005*.

Johnson, D. S., & Klug, A. (1984). Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comp. Syst. Sci.*, *28*, 167–189.

Kifer, M., Lausen, G., & Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *J. ACM*, *42*, 741–843.

Koch, C. (2002). Query rewriting with symmetric constraints. In *Proc. of FoIKS 2002*, pp. 130–147.

Kontchakov, R., Lutz, C., Toman, D., Wolter, F., & Zakharyaschev, M. (2010). The combined approach to query answering in dl-lite. In *Proc. of KR 2010*.

Krötzsch, M., & Rudolph, S. (2007). Conjunctive queries for $\mathcal{EL}$ with composition of roles. In *Proc. of DL 2007*.

Krötzsch, M., & Rudolph, S. (2011). Extending decidable existential rules by joining acyclicity and guardedness. In *Proc. of IJCAI 2011*, pp. 963–968.

Leone, N., Manna, M., Terracina, G., & Veltri, P. (2012). Efficiently computable datalog; programs. In *Proc. of KR 2012*.

Li, L., & Horrocks, I. (2003). A software framework for matchmaking based on semantic web technology. In *Proc. of WWW 2003*.

Lutz, C., Toman, D., & Wolter, F. (2009). Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In *Proc. of IJCAI 2009*, pp. 2070–2075.

Maier, D., Mendelzon, A. O., & Sagiv, Y. (1979). Testing implications of data dependencies. *Trans. Database Syst.*, *4*(4), 455–469.

Mailharrow, D. (1998). A classification and constraint-based framework for configuration. *Artif. Intell. for Eng. Design, Anal. and Manuf.*, *12*(4), 383–397.

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

Millstein, T., Levy, A., & Friedman, M. (2000). Query containment for data integration systems. In *PODS 2000*.

Mitchell, J. C. (1983). The implication problem for functional and inclusion dependencies. *Inf. and Control*, *56*, 154–173.

Nash, A., Deutsch, A., & Remmel, J. (2006). Data exchange, data integration, and chase. Tech. rep. CS2006-0859, UCSD.

Orsi, G., & Pieris, A. (2011). Optimizing query answering under ontological constraints. *PVLDB*, *4*(11), 1004–1015.

Patel-Schneider, P. F., & Horrocks, I. (2007). A comparison of two modelling paradigms in the semantic web. *J. Web Semantics*, *5*(4), 240–250.

Pérez-Urbina, H., Motik, B., & Horrocks, I. (2010). Tractable query answering and rewriting under description logic constraints. *J. Appl. Logic*, *8*(2), 186–209.

Qian, X. (1996). Query folding. In *Proc. of ICDE 1996*, pp. 48–55.

Rabin, M. O. (1969). Decidability of second-order theories and automata on infinite trees. *Trans. Am. Math. Soc.*, *141*(1–35), 4.

Rosati, R. (2007). On conjunctive query answering in $\mathcal{EL}$. In *Proc. of DL 2007*.

Rosati, R. (2011). On the finite controllability of conjunctive query answering in databases under open-world assumption. *J. Comput. Syst. Sci.*, *77*(3), 572–594.

Savo, D. F., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Romagnoli, V., Ruzzi, M., & Stella, G. (2010). Mastro at work: Experiences on ontology-based data access. In *Proc. of Description Logics*.