Taming the Infinite Chase: Query Answering under Expressive Relational Constraints

Draft containing the full version of the KR 2008 paper.

Andrea Calì1,2, Georg Gottlob1,2, Michael Kifer3
1Computing Laboratory of Quantitative Finance
Oxford University, University of Oxford, UK
andrea.cali@comlab.ox.ac.uk
georg.gottlob@comlab.ox.ac.uk
kifer@cs.sunysb.edu
2Oxford-Man Institute of Quantitative Finance
University of Oxford, UK
3Computer Science Dept.
SUNY Stony Brook, USA

August 2008

Abstract

The chase algorithm is an important tool for query evaluation and query containment under constraints such as tuple-generating dependencies (TGD) and equality generating dependencies (EGDs). So far, most of the database research has concentrated on cases where the chase terminates; notable exceptions are Johnson and Klug’s paper about functional and inclusion dependencies, and logic-based approaches such as F-Logic Lite and DL Lite. In this paper we take a general approach, and we define and study large classes of TGDs under which the chase does not always terminate and produces an infinite result; we prove that even in this case conjunctive query evaluation, and consequently query containment, are decidable. We provide tight complexity bounds for all proposed classes of TGD.

1 Introduction

Query containment is a central problem in database theory and knowledge representation. In databases, query containment is one of the important query optimization and schema integration techniques [1, 18, 22], and in knowledge representation it has been used for object classification, schema integration, service discovery, and more [7, 20].

In practice, data stored in databases and knowledge bases satisfies various integrity constraints, and it often happens that queries that are not contained in each other in general are contained over such constrained databases. This practically relevant instance of the general problem was first studied by Johnson and Klug in [18] for functional and inclusion dependencies and later in [6].

Several additional decidability results were obtained by focusing on concrete applications. For instance, in [3], constraints specific to the entity-relational diagrams were considered, and [4] used constraints derived from a subset of F-logic [19], called F-logic Lite.

In the present paper, rather than focusing on specific logical theories, we analyze the fundamental difficulty that underlies several of the earlier approaches, such as [18, 3, 4]. They
all considered special classes of so-called tuple-generating dependencies (TGDs) and equality-
generating dependencies (EGDs), all used the technique called chase, and all faced the problem
that the chase generates infinite relations. They then dealt with the infinity in their own way.
In this paper we are tackling the problem in a much more general way, by carving out a very
large class of constraints for which the infinite chase can be tamed. Our results subsume both
the main decidability and complexity result in [18] and decidability and complexity results on
F-logic lite [19] as special cases, and are actually way more general.

A TGD $\forall \bar{X} \forall \bar{Y} \Phi(\bar{X}, \bar{Y}) \rightarrow \exists \bar{Z} \Psi(\bar{X}, \bar{Z})$ is a first-order formula, where $\Phi(\bar{X}, \bar{Y})$ and $\Psi(\bar{X}, \bar{Z})$
are conjunctions of atoms over $R$. If a TGD is not satisfied by a database instance $D$, then
its body is satisfied, but its head is not satisfied by $D$. It is possible to enforce a TGD by
modifying $D$ and adding new atoms that satisfy the head. These new atoms contain labeled
null-values at the positions of $\bar{Z}$ variables. A well-known procedure that enforces the validity
of a set of TGDs is the chase [21, 18]. The chase basically applies the TGDs again and again
until a fixed-point is reached that is guaranteed to satisfy all TGDs. The result of the chase
can be finite or infinite.

The importance of the chase as a fundamental technique for query evaluation and query
containment testing under a set $\Sigma$ of TGDs relies on the fact that the chase, both in the
finite and in the infinite case, produces a solution $U$, i.e., a relational instance that contains the
initial database instance and that satisfies all TGDs [13]. Moreover, this solution $U$ is universal,
which means that $U$ can be homomorphically mapped to all other solutions. In terms of query
evaluation, this entails, for example, that for a Boolean conjunctive query $Q$, $D \cup \Sigma \models Q$ iff
$U \models Q$ [23]. The chase has been intensively used in the area of data exchange [13, 16]. Also, note
that the chase is a form of tableau, and it has successfully applied in terminological reasoning
based on description logics [8, 27].

Several authors have recently studied data exchange and query evaluation problems for
settings where the chase always terminates and thus produces a finite solution [13]. To this
aim, restrictions for sets of TGDs, such as weak acyclicity (first introduced in [12] and heavily
utilized, e.g., in [13]), have been defined that guarantee termination for whatever input database.
However, little was known about decidable query evaluation and query containment in case of
non-terminating chases that produce an infinite result. An exception is the classical work of
Johnson and Klug [18]. They showed by a very involved proof that query containment is
decidable in case all constraints are inclusion dependencies (IDs), i.e., TGDs whose body and
head consist both of a single atom only. While useful, IDs are a very restricted class of TGDs.
What about larger classes of TGDs, and what in case EGDs are added? This is precisely the
topic of the present paper.

In Section 3, we define the notions of sets of guarded TGDs (GTGDs) and of weakly guarded
TGDs (WGTGDs). A TGD is guarded if its body contains an atom called guard that covers
all variables occurring in the body. Weakly guarded TGDs are a generalization of guarded
TGDs that require guards to cover only variables occurring at affected positions, i.e., positions
in predicates that may contain some fresh labelled nulls generated during the chase. Note that
IDs are trivially guarded TGDs. To emphasize the importance of guards, we show (Theorem 20)
that there is a fixed set $\Sigma_u$ of TGDs that contains several GTGDs and a single unguarded TGD,
such that query evaluation under $\Sigma_u$ is undecidable. However, we show that, with WGTGDs,
the (possibly infinite) result of the chase has finite treewidth (Theorem 29), and use this fact
together with well-known results about the generalized tree-model property for a short proof
that Boolean query evaluation and query containment is decidable under WGTGDs (and thus
also with GTGDs). Unfortunately, this decidability result does not allow us to derive useful
complexity bounds.

In Section 4, where we present our main results, we study the complexity of query evaluation
(and equivalently, query containment) under WGTGDs. We first show by Turing machine simulations that query evaluation under WGTGDs is \textsc{exptime}-hard, in case of a fixed set of TGDS, and \textsc{2-exptime}-hard in case the TGDS are part of the input. In search of upper bounds, let us first remark that showing that $D \cup \Sigma \models Q$ is equivalent to show that the theory $T = D \cup \Sigma \cup \{\neg Q\}$ is unsatisfiable. Unfortunately, $T$ is in general not guarded, because $Q$ isn’t, and because sets of WGTGDs are generally unguarded FO sentences (while GTGDs are in guarded FO). Therefore, we cannot (as one may think at the first glance) directly or easily use known results on guarded logics [14] to derive complexity results for query evaluation. We thus develop completely new and genuine algorithms.

A key notion we introduce is the notion of a \textit{cloud} of an atom $a$ derived by the chase. Roughly, the cloud of $a$ is the set of all atoms $b$ derived by the chase such that each argument of $b$ is either in the active domain of the input instance or among the arguments of $a$. We prove that the cloud of $a$ determines all atoms whose derivation via the chase depends (even remotely) on $a$. Since there is, up to isomorphism, only a finite number of clouds, an infinite chase can be mimicked by an alternating algorithm $T\text{check}$ that uses only a finite number of configurations.

In order to answer a query $Q$, we need to match $Q$ against the atoms successively produced by $T\text{check}$. To be able to do this, we develop the new concept of a \textit{squid decomposition} of a query. Such a decomposition is obtained by a unification of the query variables that divides the query $Q$ in two parts, a head $h(Q)$ and a remaining acyclic query $t(Q)$ whose join forest are referred to as the tentacles of $Q$. Via a nondeterministically guessed squid decomposition, we can transform the query matching problem into a tree matching problem that $T\text{check}$ can solve dynamically. An analysis of the space requirements of the $T\text{check}$ algorithm allows us to derive upper bounds that match lower bounds, and we thus obtain the following main complexity result:

\textbf{Theorem 49} Let $\Sigma$ be a set of WGTGDs, let $D$ be an instance, and let $Q$ be a Boolean conjunctive query. Determining whether $D \cup \Sigma \models Q$, or, equivalently, whether $\text{chase}(\Sigma, D) \models Q$ is \textsc{exptime} complete in case of bounded predicate arities, and even in case $\Sigma$ is fixed, and in \textsc{2-exptime} complete in general. The same completeness results hold for the problem of query containment under WGTGDs.

In Section 4.4, 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 \textsc{np}-complete, and in \textsc{ptime} in case the query is atomic. Recall that Boolean query evaluation is \textsc{np}-hard even in case of a simple database without integrity constraints [9]. Therefore, the above \textsc{np} upper bound for general Boolean queries is optimal, i.e., there cannot be a reasonable class of TGDs for which query evaluation (or query containment) is more efficient.

The complexity results in this paper, together with some immediate consequence of them, are summarized in Figure 1, where all complexity bounds are tight, and $\Sigma$ denotes the set of Datalog$^3$ rules. By “bounded width” we intend bounded treewidth or even hypertree width [15]. Notice that complexity in the case of fixed queries and fixed TGDS is the so-called \textit{data complexity}, i.e., the complexity wrt the data only, which is of particular interest in database applications. With such results, we subsume both the main decidability and \textsc{np}-complexity result in [18], and decidability and complexity results on F-logic lite [4] as special cases, and we actually are way more general.

In the present paper, we deal with query answering and query containment in the classical logic setting, i.e., under arbitrary (finite or infinite) models. Our next step will be to analyze query answering and query containment with WGTGDS under finite models, i.e, the so called
finite implication problem. Interestingly, in a recent paper [26], Rosati answered a long standing open question by Johnson and Klug by showing that for the class of inclusion dependencies, the finite implication problem coincides with the general one.

2 Preliminaries

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

2.1 Alphabets

We introduce the following pairwise disjoint sets of symbols: (i) an infinite set $\Delta$ of constants, which constitute the “normal” domain of a database schema $R$; (ii) an infinite set $\Delta_N$ of labeled nulls, which will be used as “fresh” Skolem terms; (iii) an infinite set $\Delta_V$ of variables, which are used in queries. We also assume a lexicographic order on $\Delta$ and $\Delta_N$, with every symbol in $\Delta_N$ following all symbols in $\Delta$. Sets of variables (or sequences, with abuse of notation) will be denoted as $\bar{X}$, with $\bar{X} = X_1, \ldots, X_k$ for some $k$. The notation $\exists \bar{X}$ indicates $\exists X_1 \ldots \exists X_k$, and the same holds for the quantifier $\forall$.

2.2 Relational model

We will refer to a relational schema $R$, assuming that database instances (also called databases), queries and dependencies use predicates in $R$.

A relational schema $R$ is a set of relational predicates, each with its associated arity. An instance of a relational predicate $R/n$ of arity $n$ is a set of atoms of the form $R(c_1, \ldots, c_n)$, where $c_1, \ldots, c_n \in \Delta \cup \Delta_N$. Such (ground) atoms are also called tuples or facts. An instance for the relational schema $R$ is the union of instances, one for every predicate in $R$. In the following, if not stated otherwise, database instances (or simply databases) will be considered to be constructed on $\Delta \cup \Delta_N$, if not stated otherwise.

Given an atom $a$, ground or non-ground, the domain of $a$, denoted $\text{dom}(a)$, is the set of all values (variables, constants or labelled nulls) that appear as arguments in $a$; given a set of atoms $A$, we define $\text{dom}(A) = \bigcup_{a \in A} \text{dom}(a)$; we use the same notation when $A$ is a sequence or a conjunction (e.g., the body of a query) of atoms. Given a sequence or a conjunction $\xi$ of atoms, we note with $\text{atoms}(\xi)$ the set of the atoms in $\xi$. Given an atom $a$, we denote with $\text{var}(a)$ the set of variables in $a$; if $A$ is a set, conjunction, or sequence of atoms, $\text{var}(A)$ is straightforwardly defined.

An $n$-ary conjunctive query (CQ) over a schema $R$ is a formula of the form $Q(X_1, \ldots, X_n) \leftarrow \Phi(\bar{X})$, where $Q$ is a predicate not appearing in $R$, all variables $X_1, \ldots, X_n$ appear in $\bar{X}$, and $\Phi(\bar{X})$ is a conjunction of atoms constructed with predicates from $R$. The arity of a query is the arity of its head predicate $Q$; if $Q$ has arity 0, then the query is called Boolean. In the following, in the case of Boolean queries, it will be convenient not to represent the head predicate and the conjunction among the atoms, and to represent a query as a set of atoms. In the following, if not stated otherwise, we assume that queries contain no constants; it is easily seen that every

<table>
<thead>
<tr>
<th>CQ type</th>
<th>GTGDs variable $\Sigma$</th>
<th>WGTGDs variable $\Sigma$</th>
<th>GTGDs fixed $\Sigma$</th>
<th>WGTGDs fixed $\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>2-EXPTIME</td>
<td>2-EXPTIME</td>
<td>NP</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>bounded width, fixed, and atomic</td>
<td>2-EXPTIME</td>
<td>2-EXPTIME</td>
<td>PTIME</td>
<td>EXPTIME</td>
</tr>
</tbody>
</table>

Figure 1: Summary of complexity results
instance of the problem of query answering for CQs can be turned in polynomial time into an equivalent instance of the same problem for constant-free CQs. Given a query Q, its size |Q| denotes the number of its atoms.

Given a relational schema R, a position P[k] in R is identified by a predicate P in R and a natural number k, with 1 ≤ k ≤ arity(P) (arity(P) is the arity of the predicate P): k identifies the k-th attribute of P, assuming an ordering on the attributes of every relational predicate.

Henceforth, we shall deal with database instances, or simply databases, that may contain labeled nulls in ΔN as values. When such databases are treated as first-order formulae, each labeled null is viewed as an existential variable with the same name. For instance, \( P(a, \ell_1, \ell_2, \ell_1) \), where the \( \ell_i \) are nulls in \( \Delta_N \), is treated as \( \exists \ell_1, \ell_2 P(a, \ell_1, \ell_2, \ell_1) \). If confusion does not arise, we will omit the existential quantifiers.

Database instances, or simply databases, will be constructed with values from \( \Delta \cup \Delta_N \).

By an atom we mean an atomic formula of the form \( P(a_1, \ldots, a_n) \), where \( P \) is an n-ary relation name. We will use the terms “relation name” and “predicate” interchangeably. The constants appearing in an atom \( \text{denote} \) will be denoted by \( \text{dom}(a) \). This notation extends to sets and conjunctions of atoms.

### 2.3 Homomorphisms

A mapping from one set of symbols, \( S_1 \), to another set of symbols, \( S_2 \), is a function \( \mu : S_1 \rightarrow S_2 \) defined as follows: (i) \( \emptyset \) (empty mapping) is a mapping; (ii) if \( \mu_0 \) is a mapping, then \( \mu_0 \cup \{ X \rightarrow Y \} \), where \( X \in S_1 \) and \( Y \in S_2 \) is a mapping if \( \mu_0 \) does not already contain some \( X \rightarrow Y' \) with \( Y \neq Y' \). If \( X \rightarrow Y \) is in a mapping \( \mu \), we write \( \mu(X) = Y \). A homomorphism from a set of atoms \( D_1 \) to another set of atoms \( D_2 \), both over the same relational schema \( R \), is a mapping \( \mu \) from \( \Delta \cup \Delta_N \cup \Delta_V \) to \( \Delta \cup \Delta_N \cup \Delta_V \) such that the following conditions hold: (1) if \( c \in \Delta \) then \( \mu(c) = c \); (2) if \( c \in \Delta_N \) then \( \mu(c) \in \Delta \cup \Delta_N \); (3) if the atom \( R(c_1, \ldots, c_n) \) is in \( D_1 \), then the atom \( R(\mu(c_1), \ldots, \mu(c_n)) \) is in \( D_2 \). The notion of homomorphism is naturally extended to atoms as follows. If \( f = R(c_1, \ldots, c_n) \) is an atom and \( \mu \) is a homomorphism, we define \( \mu(f) = R(\mu(c_1), \ldots, \mu(c_n)) \). For a set of atoms, \( F = \{ f_1, \ldots, f_m \} \), we define \( \mu(F) = \{ \mu(f_1), \ldots, \mu(f_m) \} \). The set of atoms \( \{ \mu(f_1), \ldots, \mu(\text{atom}(f_m)) \} \) is also called image of \( F \) w.r.t. \( \mu \). In this case, we say that \( \mu \) maps \( F \) to \( \mu(F) \). For a conjunction of atoms \( \Phi = \phi_1 \wedge \ldots \wedge \phi_n \), we use \( \mu(\Phi) \) to denote the set of atoms \( \{ \mu(\phi_1), \ldots, \mu(\phi_n) \} \).

The notion of a homomorphism serves to define the answers to conjunctive queries. The answers to a conjunctive query \( Q \) of the form \( Q(X_1, \ldots, X_n) \prec \Phi(X) \) over a database instance \( D \), denoted \( Q(D) \), are defined as follows: an atom \( \ell \in \Delta^u \) is in \( Q(D) \) iff there exists a homomorphism \( \mu \) that maps \( \Phi(X) \) to atoms of \( D \), and \( (X_1, \ldots, X_n) \) to \( \ell \) (notice that only null-free tuples, i.e., tuples made of constants in \( \Delta \), are allowed to be in the answer). For a Boolean conjunctive query \( Q \), it is said that \( Q \) has positive answer on a database \( D \) iff \( \langle \rangle \) (empty tuple) is in \( Q(D) \); it has negative answer otherwise.

### 2.4 Relational dependencies

A major issue in this work are database dependencies, which are defined over a relational schema. In the relational model, the most popular dependencies are tuple-generating dependencies (TGDs) and equality-generating dependencies (EGDs), which are generalizations of inclusion dependencies and functional dependencies, respectively.

**Definition 1.** Given a relational schema \( R \), a TGD \( \sigma \) is a first-order formula of the form \( \forall X \forall Y \Phi(X, Y) \rightarrow \exists Z \Psi(X, Z) \), where \( \Phi(X, Y) \) and \( \Psi(X, Z) \) are conjunctions of atoms over \( R \), called body and head of the TGD, and denoted \( \text{body}(\sigma) \) and \( \text{head}(\sigma) \) respectively. Such a
dependency is satisfied in a database $D$ for $\mathcal{R}$ if, whenever there is a homomorphism $h$ that maps the atoms of $\Phi(\bar{X}, \bar{Y})$ to atoms of $D$, there exists an extension $h_2$ of $h$ (i.e., $h_2 \supseteq h$) that maps the atoms of $\Psi(\bar{X}, \bar{Z})$ to atoms of $D$.

**Definition 2.** Given a relational schema $\mathcal{R}$, an EGD is a first-order formula of the form $\forall \bar{X} \Phi(\bar{X}) \rightarrow X_h = X_k$, where $\Phi(\bar{X})$ is a conjunction of atoms over $\mathcal{R}$, and $X_h, X_k \in \bar{X}$. Such a dependency is satisfied if, whenever there is a homomorphism $h$ that maps the atoms of $\Phi(\bar{X})$ to atoms of $D$, we have $h(X_h) = h(X_k)$.

To simplify the notation, we will usually omit the quantifiers in TGDs and EGDs.

### 2.5 The chase

The chase process was introduced in order to enable checking implication of dependencies [21], and later also for checking query containment [18]. Informally, the chase procedure is a process of repairing a database with respect to a set of database dependencies, so that the result of the chase satisfies the dependencies. By abuse of terminology, by “chase” we may refer either to the chase procedure or to its output. Chase works on a database through so-called TGD and EGD chase rules. TGD rules come in two flavors: oblivious and restricted.

**Oblivious TGD Chase Rule.** Consider a relational database $D$ for a schema $\mathcal{R}$, and a TGD $\sigma$ on $\mathcal{R}$ of the form $\Phi(\bar{X}, \bar{Y}) \rightarrow \exists \bar{Z} \Psi(\bar{X}, \bar{Z})$. The TGD $\sigma$ is applicable to $D$ if there is a homomorphism $h$ that maps the atoms of $\Phi(\bar{X}, \bar{Y})$ to atoms of $D$. Let $\sigma$ be applicable and $h_1$ be a homomorphism that extends $h$ as follows: for each $X_i \in \bar{X}$, $h_1(X_i) = h(X_i)$; for each $Z_j \in \bar{Z}$, $h_1(Z_j) = z_j$, where $z_j$ is a “fresh” null, i.e., $z_j \in \Delta_N$, $z_j \notin D$, and $z_j$ lexicographically follows all other labeled nulls already introduced, and it is different from them. The result of the application of the TGD $\sigma$ is the addition to $D$ of all the atomic formulae in $h_1(\Psi(\bar{X}, \bar{Z}))$ that are not already in $D$. ■

The above TGD chase rule is called “oblivious” because it “forgets” to check whether the TGD is already satisfied: it adds atoms to the database anyway. The non-oblivious, or restricted TGD rule adds atoms only if the TGD is not satisfied.

**Restricted TGD Chase Rule.** Consider a relational database $D$ for a schema $\mathcal{R}$, and a TGD $\sigma$ on $\mathcal{R}$ of the form $\Phi(\bar{X}, \bar{Y}) \rightarrow \exists \bar{Z} \Psi(\bar{X}, \bar{Z})$. The TGD $\sigma$ is applicable to $D$ if $D \not\models \sigma$ (see Definition 1), i.e., there is a homomorphism $h$ that maps the atoms of $\Phi(\bar{X}, \bar{Y})$ to tuples of $D$, and there is no extension $h_1$ of $h$ (i.e., no homomorphism such that $h_1 \supseteq h$) that maps the atoms of $\Psi(\bar{X}, \bar{Z})$ to tuples of $D$. Let $\sigma$ be applicable and $h_1$ be an extension of $h$ with the above properties; the result of the application of $\sigma$ is the addition to $D$ those atoms of $h_1(\Psi(\bar{X}, \bar{Z}))$ that are not already in $D$. ■

Thus, the only difference between the oblivious and the restricted cases is that the latter applies stricter criteria to the applicability of TGDs.

**EGD Chase Rule.** Consider a relational database $D$, and an EGD $\eta$ of the form $\Phi(\bar{X}) \rightarrow X_\ell = X_k$, where $X_\ell, X_k \in \bar{X}$. The EGD $\eta$ is applicable to $D$ if there is a homomorphism $h$ that maps the atoms of $\Phi(\bar{X}, \bar{Y})$ to atoms of $D$ and $h(X_\ell) \neq h(X_k)$. If $\eta$ is applicable and $X_\ell, X_k$ are two distinct elements of $dom(D)$, then the application of the EGD yields a hard constraint violation, which in turn causes the failure of the chase, and the halting of its computation. In such a case, the result of the chase is an inconsistent theory. If $\eta$ is applicable and its application does not make the chase fail, the result of its application is the replacement of all occurrences of $h(X_\ell)$ in all $D$ with $h(X_k)$, if $h(X_k)$ precedes $h(X_\ell)$ in the lexicographical order. If $h(X_\ell)$ precedes $h(X_k)$, we replace all occurrences of $h(X_k)$ with $h(X_\ell)$. ■

It is important to keep in mind that, in the application of the TGD rule, an atom is added to the database only if it is not in the part of the chase constructed so far, and that an application of a chase rule may change the database.
An important notion in the chase is the level of an atom. Let $D$ the initial database from which the chase is constructed. Then: (1) The atoms in $D$ are said to have level 0. (2) Suppose a TGD rule for $\Phi(\bar{X},\bar{Y}) \rightarrow \exists Z \Psi(\bar{X},Z)$ is applied at some point in the construction of $chase(D)$ and let $h, h_1$ be as in the TGD chase rule. If the atom with highest level among those in $h_1(\Phi(\bar{X},\bar{Y}))$ has level $k$, then every atom in $h_1(\Psi(\bar{X},Z))$ that is actually added is at level $k + 1$.

Let $D$ be a database and $\Sigma$ a set of TGDs and EGDs. Then, the chase of $D$ with respect to $\Sigma$, denoted $Rchase(\Sigma, D)$ (resp. $Ochase(\Sigma, D)$) if the restricted TGD chase rule (resp. the oblivious one) is used, is the (possibly infinite) database constructed by an iterative application of the chase rules as follows: (1) Apply the EGD rule exhaustively, as long as it is applicable, according to some pre-established order. (2) Apply the (either oblivious or restricted) TGD chase rule once: to choose how to apply it, let $I_1, \ldots, I_k$ be all possible images of bodies of TGDs in $\Sigma$ w.r.t. some homomorphism, and let $a_i$ be the atom with highest level in $I_i$; let us define a set $M = \{ q \in \{1, \ldots, k\} \mid level(a_q) = \min_{1 \leq i \leq k} \{level(a_i)\} \}$: among the possible applications of the TGD chase rule, choose the lexicographically first among those that are applied utilizing a mapping of the body of some TGD on $I_j$, for some $j \in M$. (3) Go to step 1. It is easy to see that, in the presence of existential variables in the head of TGDs, this database might be infinite. In this case, it is important that applications of the TGD chase rule introduce atoms with the minimum possible level, so that the construction proceeds in a breadth-first fashion.

For the moment, we put EGDs aside and deal only with TGDs.

2.6 Query answering under TGDs and the chase

We now define the notion of query answering under TGDs. A similar notion is used in data exchange [13, 16] and in query answering over incomplete data [5]. Given an incomplete database, i.e., a database that does not satisfy all the constraints in $\Sigma$, we first define the set of completions (aka repairs) of that database, which we call solutions.

**Definition 3.** Consider a relational schema $\mathcal{R}$, a set of TGDs $\Sigma$, and a database instance $D$ for $\mathcal{R}$. The set of instances $B$ such that $B \models \Sigma \cup D$, is called the set of solutions of $D$ given $\Sigma$, and is denoted by $sol(\Sigma, D)$.

**Definition 4.** Consider a relational schema $\mathcal{R}$, a set of TGDs $\Sigma$, and a database instance $D$ for $\mathcal{R}$. The answers to a conjunctive query $Q$ on $D$ given $\Sigma$, denoted $ans(Q, \Sigma, D)$, is the set of ground atoms $a$ such that for every $B \in sol(\Sigma, D)$, $a \in Q(B)$ holds.

Containment of queries over relational databases has long been considered a fundamental problem in query optimization – especially query containment under constraints such as TGDs and EGDs.

**Definition 5.** Consider a relational schema $\mathcal{R}$, a set $\Sigma$ 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 $\Sigma$, denoted $Q_1 \subseteq_\Sigma Q_2$, if for every database instance $D$ for $\mathcal{R}$ such that $D \models \Sigma$ we have $Q_1(D) \subseteq Q_2(D)$.

Query containment and answering under TGDs as defined above are closely related to the notion of chase, and very close to each other, as we explain in the following.

**Theorem 6 ([23]).** Consider a relational schema $\mathcal{R}$, a set $\Sigma$ of TGDs on $\mathcal{R}$, and an atom $a$: we have that $a \in ans(Q, \Sigma, D)$ iff $Rchase(\Sigma, D) \models a$.

This result is important, and it holds because the (possibly infinite) restricted chase is a universal solution [13], i.e., a representative of all databases in $sol(\Sigma, D)$. More formally, a
universal solution for $D$ under $\Sigma$ is a (possibly infinite) database instance $U$ such that, for every $B \in \text{sol}(\Sigma, D)$, there exists a homomorphism that maps $U$ onto $B$. In [23] it is shown that the chase constructed with respect to TGDs is defined also when it is infinite, and it is a universal solution.

Consider a relational schema $\mathcal{R}$, a set $\Sigma$ of TGDs on $\mathcal{R}$, and two queries $Q_1, Q_2$ on $\mathcal{R}$. Let $\lambda$ be the “freezing” homomorphism for $Q_1$, i.e., a homomorphism that maps every distinct variable in $Q_1$, into a distinct null in $\Delta_N$. Then we say that $\lambda(\text{head}(Q_1))$ is a database instance obtained by freezing the body of $Q_1$.

**Theorem 7.** Consider a relational schema $\mathcal{R}$, a set $\Sigma$ of TGDs on $\mathcal{R}$, and two queries $Q_1, Q_2$ on $\mathcal{R}$. We have that $Q_1 \subseteq \Sigma Q_2$ iff $\lambda(\text{head}(Q_1)) \in Q_2(\text{Rchase}(\Sigma, \lambda(\text{body}(Q_1))))$, where $\lambda$ is a freezing homomorphism for $Q_1$.

From the previous results, straightforwardly obtainable from [18, 23], we easily conclude the following well-known result.

**Corollary 8.** Under TGDs, the (decision) problem of query answering on incomplete data, and the problem of query containment are mutually PTIME-reducible.

### 2.7 Oblivious vs. restricted chase

As observed in [18] in the case of functional and inclusion dependencies, things become more complicated if the restricted chase is used in place of the oblivious, since applicability of a TGD depends on the presence of other atoms previously added to the database by the chase. Indeed, the restricted chase of a database $D$ with respect to a set of TGDs $\Sigma$ is universal for $D$ under $\Sigma$, i.e., there exists a homomorphism from the restricted chase to every solution, including the oblivious chase. However, it is technically much easier to use the oblivious chase, and it can be used in lieu of the restricted chase because, as we shall prove now, the oblivious chase is also universal. This result has the status of folklore in the literature, but to the best of our knowledge has never been explicitly stated.

**Theorem 9.** Consider a set $\Sigma$ of TGDs on a relational schema $\mathcal{R}$, and let $D$ be a database on $\mathcal{R}$. Then there exists a homomorphism $\mu$ such that $\mu(\text{Ochase}(\Sigma, D)) \subseteq \text{Rchase}(\Sigma, D)$.

**Proof.** The proof goes by induction on the number $m$ of applications of the TGD chase rule, in the construction of the oblivious chase $\text{Ochase}(D, \Sigma)$. Let us consider the initial segment of the (oblivious) chase, obtained starting from $D$ and by applying $m$ times the TGD chase rule, and let us denote it with $\text{Ochase}^m(\Sigma, D)$ to $\text{Rchase}(D, \Sigma)$. We want to prove that for all $m$ with $m \geq 0$ we have that there exists a homomorphism from $\text{Ochase}^m(\Sigma, D)$ to $\text{Rchase}(D, \Sigma)$.

**Base case.** In the base case, where $m = 0$, no TGD rule has yet been applied, therefore $\text{Ochase}^0(\Sigma, D) = D \subseteq \text{Rchase}(D, \Sigma)$, so the existence of a homomorphism from $\text{Ochase}(D, \Sigma)$ to $\text{Rchase}(D, \Sigma)$ is witnessed by the identity homomorphism $\mu_0$. **Inductive case.** Assume we have applied $m$ times the TGD chase rule, obtaining an initial segment of the (oblivious) chase, that we denote with $\text{Ochase}^m(\Sigma, D)$. Now, by the induction hypothesis, there exists a homomorphism $\mu_m$ that maps $\text{Ochase}^m(\Sigma, D)$ into $\text{Rchase}(D, \Sigma)$. Consider the $(m+1)$-th application of the TGD chase rule, for the TGD: $\Phi(X, Y) \rightarrow \exists Z \Psi(X, Z)$. This means, by definition of applicability of a TGD, that there is a homomorphism $\lambda_0$ that maps $\Phi(X, Y)$ to atoms of $\text{Ochase}(D, \Sigma)$; as a consequence, $\lambda_0$ is suitably extended to $\lambda'_0$, according to the TGD chase rule, so that $\lambda'_0$ maps each of the variables in $Z$ to a fresh null in $\Delta_N$, not already present in $\text{Ochase}^m(\Sigma, D)$; then, all atoms in $\lambda'_0(\Psi(X, Z))$ are added to $\text{Ochase}^m(\Sigma, D)$, thus obtaining $\text{Ochase}^{m+1}(\Sigma, D)$. We note that there also exists another homomorphism $\lambda_R$ that
maps \( \Phi(\bar{X}, \bar{Y}) \) to atoms of \( R_{chase}(D, \Sigma) \); in particular, \( \lambda_R = \lambda_O \circ \mu_m \). Since \( R_{chase}(D, \Sigma) \) satisfies all the dependencies in \( \Sigma \) (and so does \( O_{chase}(D, \Sigma) \)), there is an extension \( \lambda'_R \) of \( \lambda_R \) that maps \( \Psi(\bar{X}, \bar{Z}) \) to tuples of \( R_{chase}(D, \Sigma) \). Denoting \( \bar{Z} = Z_1, \ldots, Z_k \), we now define

\[
\mu_{m+1} = \mu_m \cup \{ \lambda'_O(Z_i) \rightarrow \lambda'_R(Z_i) \}_{1 \leq i \leq k}
\]

To complete the proof, we now need to show that \( \mu_{m+1} \) is indeed a homomorphism. The addition of \( \lambda'_O(Z_i) \rightarrow \lambda'_R(Z_i) \), with \( 1 \leq i \leq k \), is compatible with \( \mu_m \), because none of the \( \lambda'_O(Z_i) \) appears in \( \mu_m \); therefore \( \mu_{m+1} \) is a well-defined mapping. Now, consider a generic atom \( R(\bar{X}, \bar{Z}) \) in \( \Psi(\bar{X}, \bar{Z}) \); \( \lambda'_O(R(\bar{X}, \bar{Z})) \) is the (single) atom added to \( O_{chase}(D, \Sigma) \) in the \( (m+1) \)-th step; notice that:

\[
\mu_{m+1}(R(\bar{X}, \bar{Y})) = \mu_{m+1}(R(\lambda'_O(\bar{X}), \lambda'_O(\bar{Z}))) = R(\mu_{m+1}(\lambda'_O(\bar{X})), \mu_{m+1}(\lambda'_O(\bar{Z})))
\]

Also, notice that \( \mu_{m+1}(\lambda'_O(\bar{X})) = \mu_{m+1}(\lambda_O(\bar{X})) = \lambda_R(\bar{X}) = \lambda'_R(\bar{X}) \), and \( \mu_{m+1}(\lambda'_O(\bar{Z})) = \lambda'_R(\bar{Z}) \). Therefore,

\[
\mu_{m+1}(R(\bar{X}, \bar{Z})) = R(\lambda'_R(\bar{X}), \lambda'_R(\bar{Z})) = \lambda'_R(R(\bar{X}, \bar{Z}))
\]

which is in \( R_{chase}(D, \Sigma) \) by construction.

The desired homomorphism from \( O_{chase}(\Sigma, D) \) to \( R_{chase}(\Sigma, D) \) is eventually \( \mu = \bigcup_{i=0}^{\infty} \mu_i \).

\( \square \)

**Corollary 10.** Given a set \( \Sigma \) of TGDs on a relational schema \( \mathcal{R} \) and a database \( D \) for \( \mathcal{R} \), we have that \( O_{chase}(\Sigma, D) \) is a universal solution for \( D \) under \( \Sigma \).

**Corollary 11.** Given a Boolean query \( Q \) over a schema \( \mathcal{R} \), a database \( D \) for \( \mathcal{R} \), and a set of TGDs \( \Sigma \), we have that \( O_{chase}(\Sigma, D) \models Q \) if and only if \( R_{chase}(\Sigma, D) \models Q \).

In the following, unless explicitly stated otherwise, “chase” will mean oblivious chase, and \( chase(\Sigma, D) \) will stand for \( O_{chase}(\Sigma, D) \).

**Decision problems** Recall that, by Theorem 6, \( D \cup \Sigma \models Q \) iff \( chase(\Sigma, D) \models Q \). Based on this, we define two relevant decision problems and prove their logspace-equivalence.

**Definition 12.** The conjunctive query evaluation decision problem \( CQ\text{-Eval} \) is defined as follows. Given a conjunctive query \( Q \), a set of TGDs \( \Sigma \), a database \( D \) and a ground atom \( a \), decide whether \( a \in Q(chase(\Sigma, D)) \).

**Definition 13.** The Boolean conjunctive query evaluation problem \( BCQ\text{-Eval} \) is defined as follows. Given a Boolean conjunctive query \( Q \), a set of TGDs \( \Sigma \), and a database \( D \), decide whether \( chase(\Sigma, D) \models Q \).

The following result is implicit in [9].

**Lemma 14.** The problems \( CQ\text{-Eval} \) and \( BCQ\text{-Eval} \) are logspace-equivalent.

**Proof.** Notice that \( BCQ\text{-Eval} \) can be trivially made into a special instance of \( CQ\text{-Eval} \), e.g., by adding a propositional atom as head atom. It thus suffices to show that \( CQ\text{-eval} \) polynomially reduces to \( BCQ\text{-eval} \). Let \( (Q, D, \Sigma, a) \) be an instance of \( CQ\text{-Eval} \). Assume the head atom of \( Q \) is \( R_h(X_1, \ldots, X_k) \) and \( a = R_o(c_1, \ldots, c_k) \). Then, define \( Q' \) to be the Boolean conjunctive query whose atoms are those in \( body(Q) \) plus \( R_h^+(X_1, \ldots, X_k) \), where \( R_h^+ \) is a fresh predicate symbol not occurring in \( D \) and \( Q \) (and therefore not in \( \Sigma \), since the TGDs in \( \Sigma \) are on the same relational schema). Moreover, let \( a' = R_h^+(c_1, \ldots, c_k) \). It is easy to see that \( a \in Q(chase(\Sigma, D)) \) iff \( chase(\Sigma, D \cup \{a'\}) \models Q' \).

\( \square \)
By the above lemma, and by the well-known equivalence of the problem of query containment under TGDs with the CQ-eval problem (Corollary 8), the three following problems are Logspace-equivalent: (1) CQ-eval under TGDs, (2) BCQ-eval under TGDs, (3) query containment under TGDs. Henceforth, we will concentrate on only one of these problems, namely the BCQ-eval problem. All complexity results carry over to the other problems.

2.8 A more convenient form of TGDs

Before presenting our main results, we prove the following lemma, which will allow us to simplify the proofs in the rest of the paper. The lemma states that we can restrict our attention on TGDs that have singleton atoms in the head.

Lemma 15. Let $Q$ be a conjunctive query over a schema $\mathcal{R}$, and let $\Sigma$ be a set of TGDs on $\mathcal{R}$. Then, from $\Sigma$ we can construct in Logspace a set of TGDs $\Sigma'$ such that each TGD in $\Sigma'$ has only one atom in its head and for each instance $D$, chase($\Sigma$, $D$) $\models$ $Q$ iff chase($\Sigma'$, $D$) $\models$ $Q$.

Proof. To obtain $\Sigma'$ from $\Sigma$ is sufficient to replace each rule of the form $r : \text{body}(\bar{X}) \rightarrow \text{head}_1(\bar{Y}), \text{head}_2(\bar{Y}), \ldots, \text{head}_k(\bar{Y})$, where $k > 1$ and $\bar{Y}$ is the set of all the variables that appear in the head (that may include part of $\bar{X}$), with the following set of rules:

\[
\begin{align*}
\text{body}(\bar{X}) & \rightarrow V(\bar{Y}) \\
V(\bar{Y}) & \rightarrow \text{head}_1(\bar{Y}) \\
V(\bar{Y}) & \rightarrow \text{head}_2(\bar{Y}) \\
\vdots \\
V(\bar{Y}) & \rightarrow \text{head}_k(\bar{Y}), 
\end{align*}
\]

where $V$ is a fresh predicate symbol, having the same arity as the number of variables in $\bar{Y}$; notice also that in general not all the variables in $\bar{Y}$ also appear in $\bar{X}$. It is straightforward to see that, except for the atoms of the form $V(\bar{Y})$, chase($\Sigma$, $D$) and chase($\Sigma'$, $D$) coincide. The atoms of the form $V(\bar{Y})$, being introduced only in the transformation above, do not match any predicate symbol in $Q$, hence, chase($\Sigma$, $D$) $\models$ $Q$ iff chase($\Sigma'$, $D$) $\models$ $Q$. \(\square\)

Therefore, we can always assume without loss of generality that each TGD in the set $\Sigma$ has a singleton atom in its head.

2.9 Tree decomposition and related notions

We now give some preliminary notions about tree decompositions. A hypergraph is a pair $\mathcal{H} = (V, H)$, where every $h \in H$ is called hyperedge, and it is a subset of $V$. The Gaifman graph of a hypergraph $\mathcal{H}$, denoted $\mathcal{G}_\mathcal{H}$, is a graph having the same $V$ as set of nodes, and such that there is an edge $(v_1, v_2)$ if $v_1$ and $v_2$ both occur in the same hyperedge in $H$.

Given a graph $\mathcal{G} = (V, E)$, a tree decomposition of $\mathcal{G}$ is a pair $(T, \lambda)$, where $T = (N, A)$ is a tree, and $\lambda$ a labeling function $\lambda : N \rightarrow 2^V$ such that: (i) for all $v \in V$, there exists $n \in N$ such that $v \in \lambda(n)$; more briefly, $\lambda(N) = V$ (where $\lambda(N)$ denotes $\bigcup_{n \in N} \lambda(n)$); (ii) for every arc $e \in E$, with $e = (v_1, v_2)$, there exists $n \in N$ 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 $(T, \lambda)$ is the integer value $\max\{|\lambda(n)| - 1 \mid n \in N\}$. The treewidth of a graph $\mathcal{G} = (V, E)$, denoted $tw(\mathcal{G})$, is the minimum width of all tree decompositions. Given a hypergraph $\mathcal{H}$, its treewidth $tw(\mathcal{H})$ is defined as the treewidth of its Gaifman...
graph: $tw(H) = tw(G_H)$. Notice that the notion of treewidth immediately extends to structures; therefore, since we can see database instances and queries as structures, the treewidth of databases and queries is defined.

A class $C$ of logical formulae has the bounded-treewidth model property if there exists a $k$ such that, whenever a formula $\phi \in C$ is decidable, then $\phi$ has a model of treewidth at most $k$.

The following result is straightforwardly follows from \[14\].

**Theorem 16.** If a set of first-order formulae has the bounded-treewidth model property, then checking satisfiability for it is decidable.

### 3 (Weakly) Guarded TGDs

This section introduces a special class of guarded TGDs, which have a number of useful properties.

We first give the notion of affected position of a relational schema, given a set of TGDs $\Sigma$.

**Definition 17.** Given a relational schema $R$ and a set of TGDs $\Sigma$ over $R$, a position $\pi_h$ in the predicate of the head atom of a TGD $\sigma$ in $\Sigma$ is affected with respect to $\Sigma$ if either:

- an existentially quantified variable appears in $\pi_h$, or
- the variable appearing in $\pi_h$ also appears in the in the body of $\sigma$, and only in affected positions.

**Example 1.** Consider the following set of TGDs:

$$
\begin{align*}
\sigma_1 : & P_1(X,Y), P_2(X,Y) \rightarrow \exists Z P_2(Y,Z) \\
\sigma_2 : & P_2(X,Y), P_2(W,X) \rightarrow P_1(Y,X)
\end{align*}
$$

Notice that $P_2[2]$ is affected since $Z$ in $\sigma_1$ is existentially quantified in $\sigma_1$. Considering again $\sigma_1$, the variable $Y$ appears in $P_2[2]$ but also in $P_1[2]$, therefore it does not make the position $P_2[2]$ affected. In $\sigma_2$, $X$ appears in the affected position $P_2[2]$ but also in $P_2[1]$, which is not affected; therefore, it does not make the position $P_1[2]$ affected. On the contrary, in $\sigma_2$, $Y$ appears in $P_2[2]$ and nowhere else, thus causing $P_1[1]$ to be affected. \[\square\]

**Definition 18.** Given a TGD $\sigma$ of the form $\Phi(\bar{X}, \bar{Y}) \rightarrow \Psi(\bar{X}, \bar{Z})$, we say that $\sigma$ 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 $\sigma$, i.e., all the variables $\bar{X}, \bar{Y}$ that occur in $\Phi(\bar{X}, \bar{Y})$, the body of $\sigma$.

**Definition 19.** Given a TGD $\sigma$ of the form $\Phi(\bar{X}, \bar{Y}) \rightarrow \Psi(\bar{X}, \bar{Z})$, belonging to a set of TGDs $\Sigma$ on a schema $R$, we say that $\sigma$ is a weakly guarded TGD (WGTGD) if there exists an atom in the body, called a weak guard, that contains all the universally quantified variables of $\sigma$ that appear in affected positions and do not also appear in non-affected positions. (See Definition 17.)

It is important to realize that the transformation described in Lemma 15 preserves the guardedness and weak guardedness properties. Therefore, we can still assume that TGDs have singleton heads.

The following theorem shows that it is essentially unguarded rules that are responsible for the undecidability of the main problems treated in this paper. Even a single unguarded rule can destroy the decidability of simplest reasoning tasks under TGDs.

**Theorem 20.** There is a fixed set of TGDs $\Sigma_u$ such that all but one TGDs of $\Sigma_u$ are weakly-guarded and it is undecidable whether $D \cup \Sigma_u \models Q$, or, equivalently, whether $Q \in \text{chase}(\Sigma_u, D)$.\[11\]
Consider the following TGDs: the current state is done by using the infinite grid, where the behaviour of a deterministic Turing machine (TM) \( M \) with an empty input tape. 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 \( M \) are encoded into a relation \( \text{trans} \) of \( D \), where for example, the ground atom \( \text{trans}(s_1, a_1, s_2, a_2, \text{right}) \) means “if the current state is \( s \) 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 fix the initial configuration of \( M \)) the atom \( \text{index}(0) \), which fixes the initial point of the grid. Also, we make use of three constants \( \text{right}, \text{left}, \text{stay} \) for encoding the three types of moves. Consider the following TGDs:

\[
\begin{align*}
\text{index}(X) &\rightarrow \exists Y \text{ next}(X, Y) \\
\text{next}(X, Y) &\rightarrow \text{index}(Y) \\
\text{trans}(T), \text{next}(X_1, X_2), \text{next}(Y_1, Y_2) &\rightarrow \\
\text{grid}(T, X_1, Y_1, X_2, Y_2)
\end{align*}
\]

Note that only the last of these three TGDs is unguarded. 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, and where, 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 \( M \) using suitable initialization atoms in \( D \) and guarded TGDs. To this aim, we need additional predicates \( \text{cursor}(Y, X) \) (the cursor is in position \( X \) at time \( Y \)), \( \text{state}(Y, S) \) (\( M \) is in state \( S \) at time \( Y \)), \( \text{content}(X, Y, A) \) (at time \( Y \), the content of position \( X \) in the tape is \( A \)). The following rule encodes the behaviour of the TM \( M \) on all transition rules that move the cursor to the right:

\[
\begin{align*}
\text{grid}(S_1, A_1, S_2, A_2, \text{right}, X_1, Y_1, X_2, Y_2), \\
\text{cursor}(Y_1, X_1), \\
\text{state}(Y_1, S_1), \text{content}(X_1, Y_1, A_1) &\rightarrow \\
\text{cursor}(Y_2, X_2), \text{content}(X_1, Y_2, A_2), \\
\text{state}(Y_2, S_2), \text{mark}(Y_1, X_1)
\end{align*}
\]

Such a rule has also obvious sibling rules for “left” and “stay” moves.

Additional “inertia” rules are needed, that make use of the \( \text{mark} \) predicate, similarly to what is done in the proof of Theorem 30: these rules ensure that all non-marked positions in the tape are not modified. To do this, at every instant \( Y_1 \), we adopt two different markings: one for the tape positions that follow the one marked with \( \text{mark} \), and one for those that precede it. In this way, we are able to make use of guarded rules only.

\[
\begin{align*}
\text{mark}(Y_1, X_1), \text{grid}(T, X_1, Y_1, X_2, Y_2) &\rightarrow \text{mark}_f(Y_1, X_2) \\
\text{mark}_f(Y_1, X_1), \text{grid}(T, X_1, Y_1, X_2, Y_2) &\rightarrow \text{mark}_f(Y_1, X_2) \\
\text{mark}(Y_1, X_2), \text{grid}(T, X_1, Y_1, X_2, Y_2) &\rightarrow \text{mark}_p(Y_1, X_1) \\
\text{mark}_p(Y_1, X_2), \text{grid}(T, X_1, Y_1, X_2, Y_2) &\rightarrow \text{mark}_p(Y_1, X_1)
\end{align*}
\]

The actual inertia rules follow.

\[
\begin{align*}
\text{mark}_f(Y_1, X_1), \text{grid}(T, X_1, Y_1, X_2, Y_2), \text{content}(X_1, Y_1, a) &\rightarrow \text{content}(X_1, Y_2, a) \\
\text{mark}_p(Y_1, X_1), \text{grid}(T, X_1, Y_1, X_2, Y_2), \text{content}(X_1, Y_1, a) &\rightarrow \text{content}(X_1, Y_2, a)
\end{align*}
\]
Notice that $a$ is a generic symbol in the tape alphabet $a_1, \ldots, a_\ell, b$; therefore, we write two rules as above for every tape symbol, having $2\ell + 2$ inertia rules altogether. Notice that the fact that the rules have multiple atoms in the head is not a loss of generality by virtue of Lemma 15. Finally, we assume without loss of generality that our Turing machine $M$ has a single halting state $s_0$ which is encoded by the atom $halt(s_0)$ in $D$. We add a guarded rule

$$state(Y, S), halt(S) \rightarrow stop$$

It is now clear that the machine halts iff $chase(\Sigma_u, D) \models stop$, i.e., iff $\Sigma_u \cup D \models stop$. We have thus reduced the halting problem to the problem of answering atomic queries to a database under $\Sigma_u$. The latter problem is thus undecidable. \hfill \Box

**Definition 21.** Let $\Sigma$ be a set of WGTGDs, $D$ be a database, and $U = chase(\Sigma, D)$. The guarded chase graph $GCG(\Sigma, D)$ is defined as follows. The set of vertices is constituted by the atoms of $U$, and there are two kinds of arcs: normal and dotted. Consider a TGD $\rho$ with a weak guard $\gamma \in \text{body}(\rho)$, which was used in an application of a TGD rule using a homomorphism $h$ that maps the body and head of $\rho$ to $chase(\Sigma, D)$. Then (i) there is a dotted arc from every atom in $h(\text{body}(\rho)) - \{h(\gamma)\}$ to every atom in $h(\text{head}(\rho))$; (ii) there is a normal arc from $h(\gamma)$ to every atom of $h(\text{head}(\rho))$.

**Lemma 22.** The graph obtained from $GCG(\Sigma, D)$ by omitting all dotted arcs is a forest.

**Proof.** Follows directly from the definition. \hfill \Box

**Definition 23.** The forest obtained from $GCG(\Sigma, D)$ by dropping all dotted arcs is called the guarded chase forest and is denoted by $GCF(\Sigma, D)$.

**Definition 24.** Let $D$ be a possibly infinite relational structure over $\Delta \cup \Delta_N \cup \Delta_V$ for a schema $\mathcal{R}$, and let $S$ be a set of symbols such that $S \subseteq \text{dom}(D)$.

- An $[S]$-join forest of $D$ is an undirected labeled forest $T = (V, E, \lambda)$, whose labeling function $\lambda : V \rightarrow D$ is such that:
  
  (i) $D \subseteq \lambda(V)$, and
  
  (ii) $T$ is $[S]$-connected, i.e., for each $c \in \text{dom}(D) - S$, the set $\{v \in V \mid c \text{ occurs in } \lambda(v)\}$ induces a connected subtree in $T$.

- We say that $D$ is $[S]$-acyclic iff $D$ has an $[S]$-join forest.

The above definition generalizes the classical notion of hypergraph acyclicity [2] of an instance (or, equivalently, of a query). In fact, an instance or a query (seen as an instance) is hypergraph-acyclic iff it is $[\emptyset]$-acyclic.

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

**Lemma 25.** Given a database instance $D$ for a schema $\mathcal{R}$, and a set $S$, if $D$ is $[S]$-acyclic, then $tw(D) \leq |S| + w$, where $w$ is the maximum arity of any predicate symbol in $\mathcal{R}$.

**Proof (sketch).** A tree decomposition $T = (V, E, \chi)$ of width $\leq |S| + w$ can be obtained from an $[S]$-join tree $T = (V, E, \lambda)$ of $B$ by defining $\forall v \in V, \chi(v) = S \cup \lambda(v)$.

**Definition 26.** Let $D$ be an instance for a schema $\mathcal{R}$. The Herbrand Base $HB(D)$ of $D$ is the set of all atoms that can be formed using the predicate symbols of $\mathcal{R}$ and arguments in $\text{dom}(D)$. We define:
\begin{itemize}
\item $\text{chase}^\perp(\Sigma, D) = \text{chase}(\Sigma, D) \cap \text{HB}(D)$, and
\item $\text{chase}^\wedge(\Sigma, D) = \text{chase}(\Sigma, D) - \text{chase}^\perp(\Sigma, D)$
\end{itemize}

Notice that $\text{chase}^\perp(\Sigma, D) \cup \text{chase}^\wedge(\Sigma, D) = \text{chase}(\Sigma, D)$ and $\text{chase}^\perp(\Sigma, D) \cap \text{chase}^\wedge(\Sigma, D) = \emptyset$. Moreover, if $D$ is null-free (which will be the case in many applications), then $\text{chase}^\wedge(\Sigma, D)$ is the finite set of all null-free atoms in $\text{chase}(\Sigma, D)$, while $\text{chase}^\perp(\Sigma, D)$ may be infinite.

**Lemma 27.** If $\Sigma$ is a set of WGTGDs and $D$ an instance, then $\text{chase}^\wedge(\Sigma, D)$ is $[\text{dom}(D)]$-acyclic.

**Proof.** Construct a $[\text{dom}(D)]$-join forest of $U = \text{chase}(\Sigma, D)$ as follows. The forest is simply the guarded chase forest $(U, F) = \text{GCF}(\Sigma, D)$, extended by a labeling function $\lambda$ that labels each atom by itself, i.e., $\forall u \in U, \lambda(u) = u$. We have to show that this forest is $[\text{dom}(D)]$-connected.

Assume it is not. Then, there exists an element $x \in \text{dom}(U) - \text{dom}(D)$ that appears in two vertices $v$ and $w$ of the forest $(U, F)$ but does not appear on the unique path between these two vertices. Follow the ancestors of $v$ in the guarded chase forest $\text{GCF}(\Sigma, D)$ as long as $x$ appears in their label, and let the highest among such ancestors be $v'$ ($x$ thus appears in all vertices on the path from $v$ to $v'$). Similarly, $w'$ denotes the highest ancestor of $w$ such that $x$ appears on the path from $w$ to $w'$. Clearly, $v' \neq w'$ for otherwise there would be a path between $v$ and $w$, all of whose vertices are labeled by $x$. Given that we assumed (w.l.o.g.) that all TGDs in $\Sigma$ are single headed, $v'$ and $w'$ must have been generated in different chase steps, and $x$ must be a new variable generated in each of these steps. However, according to the definition of the chase, the new variables in each step are fresh (newly introduced) and cannot coincide. Thus $(U, F)$ is $[\text{dom}(D)]$-connected. It immediately follows that $\text{chase}^\wedge(\Sigma, D)$ is $[\text{dom}(D)]$-connected.

**Lemma 28.** If $\Sigma$ is a set of WGTGDs and $D$ an instance of a schema $\mathcal{R}$, then $\text{tw}(\text{chase}(\Sigma, D)) \leq |\text{dom}(D)| + w$, where $w$ is the maximum arity of a predicate in $\mathcal{R}$.

**Proof.** The lemma is easily obtained by combining the proofs of Lemma 25 and Lemma 27, and by taking as tree decomposition precisely the one described in the proof of Lemma 27.

**Theorem 29.** Given a relational schema $\mathcal{R}$, a set of WGTGDs $\Sigma$, a conjunctive query $Q$, and a database instance for $\mathcal{R}$, the problem of checking $\text{chase}(\Sigma, D) \models Q$ is decidable.

**Proof.** We rely on the fact that both $\text{chase}(\Sigma, D) \land Q$ and $\text{chase}(\Sigma, D) \land \neg Q$ have a (possibly infinite) model of finite treewidth, when they are satisfiable. This follows from the fact that $\text{chase}(\Sigma, D)$ is universal for $D$ under $\Sigma$ and has finite treewidth. Our claim now follows by a well-known result of Courcelle [10], that generalizes an earlier result of Rabin [25]. This result states that classes of first-order logic (more generally, monadic second-order logic) that enjoy the finite treewidth model property are decidable. A class $\mathcal{C}$ of formulae has the finite-treewidth model property if for each $\phi \in \mathcal{C}$, whenever $\phi$ is satisfiable, then it is possible to compute a number $f(\phi)$ such that $\phi$ has a model of treewidth $\leq f(\phi)$ (see also [14], where a more general property called the generalized tree-model property is discussed).

The above theorem establishes decidability of query answering under WGTGDs, but it tells nothing about the complexity. This is the subject of the next section.

### 4 Complexity

In this section we present several complexity results about query answering under guarded and weakly-guarded TGDs.
4.1 EXPTIME Hardness

**Theorem 30.** Given a relational schema $R$, a set of WGTGDs $\Sigma$, a conjunctive query $Q$, and a database instance for $R$, the problem of determining whether $\text{chase}(\Sigma, D) \models Q$ is EXPTIME-hard. In the case where the arity of predicates in $R$ is not fixed, the same problem is 2-EXPTIME hard.

**Proof.** First of all, it is well-known that $\text{APSPACE}$ (alternating $\text{pspace}$) equals EXPTIME. Notice that alternating $\text{LINSPACE}$ is already EXPTIME-hard, so to prove our claim it suffices to simulate the behavior of an Alternating Turing Machine (ATM) $M$ on an input $I$ (that will be a bit-string) by means of a set of WGTGDs. Namely, we will show that $M$ accepts the input $I$ iff $\text{chase}(\Sigma, D) \models Q$.

Without loss of generality, we can assume that the ATM $M$ has exactly one accepting (halting) state, which we denote with $s_f$. We also assume that $M$ never tries to read beyond its tape boundaries. Let $M$ be defined as

$$M = (S, \Lambda, b, \delta, s_0, F)$$

where $S$ is the set of states, $\Lambda$ is the tape alphabet (assumed to be $\{0, 1, b\}$), $b$ is the blank tape symbol, $\delta$ is the transition function, defined as $\delta: S \times \Lambda \rightarrow (S \times \Lambda \times \{\ell, r, \bot\})^2$ ($\bot$ denotes the “stay” head move, while $\ell$ and $r$ denote “left” and “right” respectively), $s_0 \in S$ is the initial state, and $F \subseteq S$ is the set of final states. Being $M$ an alternating Turing machine (ATM), the set of states $S$ is partitioned into two sets $S_\exists$ and $S_\forall$ (universal and existential states, respectively). The general idea of the encoding is that configurations of $M$ will be represented by the fresh constants that are generated in the construction of the chase. In particular, a special constant $\kappa$ will represent the initial configuration, while $v_i$, where $i \geq 1$, will denote the fresh constants 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 $\text{symbol}(a, c, v)$ denotes that in configuration $v$ the cell $c$ contains the symbol $a$, with $a \in \Lambda$. Also, a binary predicate $\text{succ}(c_1, c_2)$ denotes the fact that cell $c_1$ follows cell $c_2$ on the tape. Finally, $\text{neq}(c_1, c_2)$ says that two cells are distinct.

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

3. **Configurations.** A unary predicate $\text{config}(c)$ expresses the fact that the constant $c$ is a valid configuration. A ternary predicate $\text{next}(v, v_1, v_2)$ 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 $\text{init}(v)$ states that the configuration $v$ is initial.

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

5. **Marking.** We use $\text{mark}(c, v)$ 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 whole transition function \( \delta \) of an ATM, 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 instance \( D \). We construct a database out of distinct (and possibly infinite) alphabets of cells, configurations, and states. For brevity, we will not specify to which alphabet each constant belongs, since this will be obvious from the context. In particular, we will use an accepting state \( s_a \) and an initial state \( s_0 \), and a special initial configuration \( \kappa \). The database describes the initial configuration of the ATM, plus some technicalities.

(a) We consider the input \( I \) which, without loss of generality, are assumed to occupy the cells numbered from 1 to \( n \), i.e., \( c_1, \ldots, c_n \). Therefore, for the \( i \)-th cell of \( I \) containing the tape symbol \( a \), the database has the fact symbol\((a,c_i,\kappa)\).

(b) An atom state\((s_0,\kappa)\) specifies that \( M \) starts in state \( s_0 \) and it is the initial configuration.

(c) For every existential state \( s_E \) and a universal state \( s_U \), we have the facts existential\((s_E)\) and universal\((s_U)\). For the accepting state, the database has the fact accept\((s_a)\).

(d) An atom cursor\((c_1,\kappa)\) indicates that, in the initial configuration, the cursor points at the first cell.

(e) The atoms succ\((c_1,c_2), \ldots, succ(c_{n-1},c_n)\) encode the fact that the cells \( c_1, \ldots, c_n \) are adjacent. Also, atoms of the form neq\((c_i,c_j)\), for \( 1 \leq i, j \leq n \), with \( i \neq j \), denote the fact that the cells \( c_1, \ldots, c_n \) are pairwise distinct.

(f) The atom config\((\kappa)\) says that \( \kappa \) is a valid configuration.

(g) The database has the atoms of the form transition\((s,a,s_1,a_1,m_1,s_2,a_2,m_2)\), which encode the transition function \( \delta \), as described above.

The TGDs. Once the database is set, we are ready to describe the TGDs that define the transitions between configurations and the accepting configurations of the ATM.

(a) Configuration generation. The following TGDs say that, for every state (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:

\[
\text{config}(V), \text{state}(S,V) \rightarrow \text{next}(V,V_1,V_2)
\]

\[
\begin{align*}
\text{next}(V,V_1,V_2) & \rightarrow \text{config}(V_1), \text{config}(V_2) \\
\text{next}(V,V_1,V_2) & \rightarrow \text{follows}(V,V_1) \\
\text{next}(V,V_1,V_2) & \rightarrow \text{follows}(V,V_2)
\end{align*}
\]

(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), \( M_1, M_2 \) represent the two
moves, and the constants \( r \) and \( \ell \) represent the “right” and the “left” moves, respectively.

\[
\text{transition}(S, A, S_1, A_1, M_1, S_2, A_2, M_2), \\
M_1 = r, M_2 = \ell, \text{next}(V, V_1, V_2), \\
\text{state}(S, V), \text{cursor}(C, V), \text{symbol}(A, C, V), \\
\text{succ}(C_1, C), \text{succ}(C, C_2) \rightarrow \\
\text{state}(S_1, V_1), \text{state}(S_2, V_2), \\
\text{symbol}(A_1, C_1, V_1), \text{symbol}(A_2, C_2, V_2), \\
\text{cursor}(C_1, V_1), \text{cursor}(C_2, V_2), \text{mark}(C, V),
\]

The other eight kinds of moves of the ATM are encoded by analogous TGDs. The above rule (and its seven siblings) suitably mark the cells that are written by the transition by means of the predicate \( \text{mark} \). The cells that are not involved in the transition must retain their symbols, which is specified by the next TGD:

\[
\text{config}(V), \text{follows}(V_1, V), \text{mark}(C, V), \text{symbol}(C_1, A, V), \text{neq}(C_1, C) \rightarrow \text{symbol}(C_1, A, V_1)
\]

(c) Termination. The meaning of the following rule is clear:

\[
\text{state}(s_a, V) \rightarrow \text{accept}(V)
\]

The following TGDs state that, for existential states, at least one configuration derived from it must be accepting. For universal states, both configurations must be accepting.

\[
\text{next}(V, V_1, V_2), \text{state}(S, V), \text{existential}(S), \text{accept}(V_1) \rightarrow \text{accept}(V) \\
\text{next}(V, V_1, V_2), \text{state}(S, V), \text{existential}(S), \text{accept}(V_2) \rightarrow \text{accept}(V) \\
\text{next}(V, V_1, V_2), \text{state}(S, V), \text{universal}(S), \text{accept}(V_1), \text{accept}(V_2) \rightarrow \text{accept}(V)
\]

Notice that the above TGDs often have more than one atom in the head, but it is always possible to replace them with sets of TGDs that have only one predicate in the head. Our construction uses multiple heads to shorten the specification and for more clarity.

Now 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 \( \text{chase}(\Sigma, D) \models \text{accept}(\kappa) \). It also easy to verify that all TGDs are WGTGDs. This proves the claim.

In the case where the arity of predicates in \( \mathcal{R} \) is not fixed, instead of simulating a \( \text{LINSPACE} \) ATM, we are able to simulate \( 2^n \) tape cells, where \( n \) is the length of the input, so that we can prove that the problem in question is \( \text{ASPACE}(2^n) \)-hard. Being \( \text{ASPACE}(2^n) = 2\text{-EXPTIME} \), it immediately follows that when the arity is not bounded the problem is \( 2\text{-EXPTIME} \)-hard. \( \square \)

4.2 Squid Decompositions

In this section we 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 subsections.

**Definition 31.** Let \( Q \) be a Boolean conjunctive query over a database schema \( \mathcal{R} \), where \( Q \) has \( n \) (body-) atoms. An \( \mathcal{R} \)-cover of \( Q \) is a Boolean conjunctive query \( Q^+ \) over \( \mathcal{R} \) that contains in its body all atoms of \( Q \) and that may, in addition, contain at most \( n \) further \( \mathcal{R} \)-atoms whose variables can be either from \( \text{var}(Q) \) or new.
Example 2. Let \( \mathcal{R} = \{ R/2, S/3, T/3 \} \), and let \( Q \) be the Boolean conjunctive query
\[ R(X, Y), R(Y, Z), T(Z, X, X). \]
The following query \( Q^+ \) is an \( \mathcal{R} \)-cover of \( Q \): \( Q^+ = \{ R(X, Y), R(Y, Z), T(Z, X, X), Y, Z, Z, S(Z, U, U) \} \).

Lemma 32. Let \( D \) be a (finite or infinite) instance over a schema \( \mathcal{R} \) and \( Q \) a Boolean CQ
over \( D \). Then \( D \models Q \) iff there exists an \( \mathcal{R} \)-cover \( Q^+ \) of \( Q \) such that \( D \models Q^+ \).

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
\( h : varQ^+ \rightarrow dom(D) \), such that \( h(Q^+) \subseteq D \), then, given that \( Q \) is a subset of \( Q^+ \), the restriction
\( h' \) of \( h \) to \( var(Q) \) is a homomorphism \( var(Q) \rightarrow dom(D) \) such that \( h'(Q) = h(Q) \subseteq D \). \( \square \)

Definition 33.\(^1\) 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 \( h : var(Q^+) \rightarrow var(Q^+) \),
and a decomposition of \( h(Q^+) \) into two sets \( H \) and \( T \), with \( T = h(Q^+) - H \), such that:
(i) there exists \( V_\delta \subseteq var(Q^+) \) such that \( H = \{ a \in h(Q^+) \mid var(a) \subseteq V_\delta \} \); (ii) \( T \) is \( |V_\delta| \)-acyclic. We refer
to \( H \) as the head of \( \delta \), and to \( T \) as the tentacles of \( \delta \). The set of all squid decompositions of \( Q \)
is referred to as squid\((Q)\).

One may imagine the set \( H \) in a squid decomposition as the head of a squid, and a the set \( T \)
as a forest of tentacles attached to that head. Note that a squid decomposition \( \delta = (Q^+, h, H, T) \)
of \( Q \) does not necessarily define a query folding \([9, 24]\) of \( Q^+ \), because \( h \) does not need to be an endomorphism of \( Q^+ \):
in other terms, we do not require that \( h(Q^+) \subseteq Q^+ \). Of course, \( h \)
is a homomorphism.

Example 3. Consider the following Boolean conjunctive query (the schema is omitted for brevity):
\[
Q = \{ R(X, Y), R(X, Z), R(Y, Z), R(Z, V_1), R(V_1, V_2), R(V_2, V_3), R(V_3, V_4), R(V_4, V_5), R(V_1, V_6), R(V_6, V_3), R(V_3, V_7), R(Z, U_1), S(U_1, U_2), U_3, R(U_3, U_4), R(U_3, U_5), R(U_4, U_3) \}
\]

Let \( Q^+ \) be the Boolean query where we add the atom \( S(U_3, U_4, U_5) \) to the body. A possible
squid decomposition \( (Q^+, h, H, T) \) can be based on the homomorphism \( h \), where \( h(V_6) = V_2 \),
h\( (V_4) = h(V_5) = h(V_7) = V_3 \), and where \( h(\xi) = \xi \) for each other variable \( \xi \). The result of the decomposition
with \( V_\delta = \{ X, Y, Z \} \) is the query shown in Figure 2, where its join graph is depicted,
in order to better distinguish the (cyclic) head from the (acyclic) tentacles. Note that if we
eliminated the additional atom \( S(U_3, U_4, U_5) \), the original atoms \( R(U_3, U_4), R(U_3, U_5), R(U_4, U_5) \)
would form an uncovered cycle, and could therefore not appear simultaneously be part of the
tentacles, as \( T \) would then no longer be \( |V_\delta| \)-acyclic. \( \square \)

The two following lemmas are auxiliary technical results.

Lemma 34. Let \( Q \) be a Boolean CQ, 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 \( f : dom(Q) \rightarrow dom(U) \)
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) \( |W| < 2|Q| \).

\( ^1\)This definition corrects and supersedes the one given in the conference version which was appropriate
for relational schemas of arity 2 only. The present definition and the subsequent proofs work for
schemes of arbitrary arities.

18
(2) There is a Boolean CQ $Q^+$ such that $Q^+ \supseteq Q$ and $|Q^+| < 2|Q|$, and a homomorphism $g$ such that $g(Q^+) = W$ and $g$ extends $f$, i.e., $g(Q) = f(Q)$.

Proof.

Part 1. By hypothesis, $U$ is $[A]$-acyclic and $f : \text{dom}(Q) \to \text{dom}(U)$ with $f(Q) \subseteq U$. Since $U$ is $[A]$-acyclic, it has a (possibly infinite) $[A]$-join forest $T = (V, E, \lambda)$. We assume, without loss of generality, that distinct vertices $u, v$ of $T$ have different labels, i.e., $\lambda(u) \neq \lambda(v)$. Let $T_Q$ be the finite subforest of $T$ that contains all ancestors in $T$ of nodes $s$ such that $\lambda(s) \in f(Q)$. Let $F = (V', E', \lambda')$ 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 $(v, w)$ 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 element from $V'$.
- Finally, for each $v \in V'$, $\lambda'(v) = \lambda(v)$.

Let $W = \lambda(V')$. We claim that the forest $F$ is an $[A]$-join forest of $W$. In fact, $F$ satisfies the $[A]$-connectedness condition: assume for two distinct vertices $u$ and $w$ of $F$ and some element $b \in \text{dom}(U) - A$, $b \in \text{dom}(\lambda'(u)) \cap \text{dom}(\lambda'(w))$. Then, by construction of $F$, $u$ and $v$ are connected by a path in $F$. Assume now that some vertex $v$ lies on the unique shortest path between $u$ and $w$ in $F$. Then, by construction of $F$, $v$ lies on the unique shortest path between $u$ and $w$ in $T$, and since $T$ is an $[A]$-join forest, $b \in \text{dom}(\lambda(v)) = \text{dom}(\lambda'(v))$. Thus $F$ is an $[A]$-join forest of $W$. Moreover, by construction of the forest $F$, 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 of cardinality $< 2|Q|$ containing $f(Q)$ as a subset.

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 $1 \leq i \leq k$, $\xi_i$ is a fresh variable, that follows

\footnotesize{One may first be tempted to let $W := f(Q)$, 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. Note that taking $W := f(Q)$ works in case of arities $\leq 2$, however.}
Lemma 35. Let $G$ be an $[\mathcal{A}]$-acyclic instance. Let $G'$ be an instance obtained from $G$ by eliminating a set $S$ of atoms from $G$ where $\text{dom}(S) \subseteq \mathcal{A}$. Then $G'$ is $[\mathcal{A}]$-acyclic.

Proof. If $T$ is an $[\mathcal{A}]$-join forest for $G$, then an $A$-join forest $T'$ for $G$ can be straightforwardly obtained from $G$ by repeatedly eliminating each vertex $v$ from $T$ where $\lambda(v) \in S$. □

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

Lemma 36 (Squid Lemma). Let $\Sigma$ be a set of WGTGDs on a schema $\mathcal{R}$, $D$ a ground database instance for $\mathcal{R}$, and $Q$ a conjunctive query, then $\text{chase}(\Sigma, D) \models Q$ iff there is a squid decomposition $\delta = (Q^+, h, H, T) \in \text{squidd}(Q)$, such that $|Q^+| < 2|Q|$, and a homomorphism $\theta : \text{dom}(h(Q^+)) \rightarrow \text{dom}(\text{chase}(\Sigma, D))$ such that: (i) $\theta(H) \subseteq \text{chase}^+(\Sigma, D)$, and (ii) $\theta(T) \subseteq \text{chase}^+(\Sigma, D)$.

Proof. “If”. If there is a squid decomposition $\delta = (Q^+, h, H, T) \subseteq \text{chase}(\Sigma, D)$ and if there exists a homomorphism $\theta$ as described, then the composition $\theta \circ h$ is a homomorphism such that $(\theta \circ h)(Q) = \theta(h(Q)) \subseteq \text{chase}(\Sigma, D)$. Hence, $\text{chase}(\Sigma, D) \models Q$.

“Only if”. Assume $U = \text{chase}(\Sigma, D) \models Q$. Then, there exists a homomorphism $f : \text{var}(Q) \rightarrow \text{dom}(U)$ with $f(Q) \subseteq \text{chase}(\Sigma, D)$. By Lemma 27, $\text{chase}^+(\Sigma, D)$ is $[\text{dom}(D)]$-acyclic. By Lemma 34 it then follows that there exists a Boolean query $Q^+$ with $2|Q|$ atoms, such that all atoms of $Q$ are also contained in $Q^+$, and a homomorphism $g : \text{dom}(Q^+) \rightarrow \text{dom}(U)$ with $g(Q^+) \subseteq U$, such that $g(Q^+)$ is $[\text{dom}(D)]$-acyclic.

Partition $\text{var}(Q^+)$ into two sets $\text{var}^+(Q^+)$ and $\text{var}^+(Q^+)$ as follows:

- $\text{var}^+(Q^+) := \{x \in \text{var}(Q^+) \mid g(x) \in \text{dom}(D)\}$
- $\text{var}^+(Q^+) := \text{var}(Q^+) - \text{var}^+(Q^+)$.  

Define a mapping $h : \text{var}(Q^+) \rightarrow \text{var}(Q^+)$ as follows. For each $x \in \text{var}(Q^+)$, let $h(x)$ be the lexicographically first variable in the set $\{y \in \text{var}(Q^+) \mid g(y) = g(x)\}$. Let $V_\delta := h(\text{var}^+(Q^+))$. Moreover, let $H$ be the set of all those atoms $g$ of $h(Q^+)$ such that $\text{var}(g) \subseteq V_\delta = h(\text{var}^+(Q^+))$, and let $T = h(Q^+) - H$. Note that, by definition of $H$, $g(H) \subseteq \text{chase}^+(\Sigma, D)$ and by definition of $T$, $g(T) \subseteq \text{chase}^+(\Sigma, D)$. Let $\theta$ be the restriction of $g$ to $\text{dom}(h(Q^+))$. Clearly, $\theta$, $h$, $H$, 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 see this, observe that $\theta$ is, by construction, a bijection between $h(\text{dom}(Q^+))$ and $\text{dom}(\theta(Q^+))$, such that $h(Q^+)$ and $\theta(Q^+)$ are isomorphic. Therefore, in particular, $T \subseteq h(Q^+)$ is isomorphic to $\theta(T)$ via the restriction $\theta_T$ of $\theta$ to $\text{dom}(T)$. Note that since $\theta_T(T) = \theta(T)$ is obtained from the $[\text{dom}(D)]$-acyclic instance $\theta(Q^+)$ by eliminating only atoms all of whose arguments are in $\text{dom}(D)$ (namely the atoms in $\theta(H)$), by Lemma 34, $\theta_T(T)$ is itself $[\text{dom}(D)]$-acyclic, and therefore trivially also $[\text{dom}(D) \cap \text{dom}(\theta_T(T))]$-acyclic. $\theta_T$ is thus by construction a bijection $\text{dom}(T) \leftrightarrow \theta_T(\text{dom}(T)) = \text{dom}(\theta_T(T))$ such that $\theta_T(V_\delta \cap \text{dom}(T)) = \text{dom}(D) \cap \text{dom}(\theta_T(T))$. Hence the two pairs $(T, V_\delta \cap \text{dom}(T))$ and $(\theta_T(T), \text{dom}(D) \cap \text{dom}(\theta_T(T))$ are isomorphic, and therefore, given that $\theta_T(T)$ is $[\text{dom}(D) \cap \text{dom}(\theta_T(T))]$-acyclic, $T$ is $[V_\delta]$-acyclic (in fact, if an instance is $[A]$-acyclic, and if $A \subseteq B$, then it is also $[B]$-acyclic). □
4.3 Clouds and the complexity of query answering under WGTGDs

To study the complexity of query answering under WGTGDs, we introduce the notion of cloud.

**Definition 37.** Let \( \Sigma \) be a set of WGTGDs on a schema \( R \) and \( D \) an instance for \( R \). For every atom \( a \) of \( \text{chase}(\Sigma, D) \) the cloud of \( a \) with respect to \( \Sigma \) and \( D \), denoted \( \text{cloud}(\Sigma, D, a) \), is the set of all atoms in \( \text{chase}(\Sigma, D) \) whose arguments are in \( \text{dom}(a) \cup \text{dom}(D) \). More formally, \( \text{cloud}(\Sigma, D, a) = \{ b \in \text{chase}(\Sigma, D) | \text{dom}(b) \subseteq \text{dom}(a) \cup \text{dom}(D) \} \). Notice that for every atom \( a \in \text{chase}(\Sigma, D) \) we have \( D \subseteq \text{cloud}(\Sigma, D, a) \). Moreover, we define

\[
\text{clouds}(\Sigma, D) = \{ \text{cloud}(\Sigma, D, a) | a \in \text{chase}(\Sigma, D) \}
\]

\[
\text{clouds}^+(\Sigma, D) = \{ (a, \text{cloud}(\Sigma, D, a)) | a \in \text{chase}(\Sigma, D) \}
\]

A set \( S \subseteq \text{cloud}(\Sigma, D, a) \) is called a subcloud of \( a \) (with respect to \( \Sigma \) and \( D \)). The set of all subclouds of an atom \( a \) is denoted by \( \text{subclouds}(\Sigma, D, a) \). Finally, we define \( \text{subclouds}^+(\Sigma, D) = \{ (a, C) | a \in \text{chase}(\Sigma, D) \wedge C \subseteq \text{cloud}(\Sigma, D, a) \} \).

**Definition 38.** Let \( D \) be an instance for a schema \( R \). Let \( \alpha \) and \( \beta \) be two constructs consisting each of one atom of \( \text{HB}(D) \), or a set of atoms of \( \text{HB}(D) \), or an atom paired with a set of atoms of \( \text{HB}(D) \). We say that \( \alpha \) and \( \beta \) are \( D \)-isomorphic, denoted \( \alpha \simeq_D \beta \), or simply \( \alpha \simeq \beta \) in case \( D \) is understood, iff there exists a bijection (i.e., a bijective homomorphism) \( f : \text{dom}(\alpha) \rightarrow \text{dom}(\beta) \) such that \( f(\alpha) = \beta \) and for all \( c \in \text{dom}(D) \), \( f(c) = f^{-1}(c) = c \).

**Example 4.** If \( a, b \in \text{dom}(D) \) and \( \zeta_1, \zeta_2, \zeta_3, \zeta_4 \notin \text{dom}(D) \), we have: \( P(a, \zeta_1, \zeta_2) \simeq P(a, \zeta_3, \zeta_4) \) and \( (P(a, \zeta_3), \{ Q(a, \zeta_3), Q(\zeta_3, \zeta_3), R(\zeta_3) \}) \simeq (P(a, \zeta_1), \{ Q(a, \zeta_1), Q(a, \zeta_1), R(\zeta_1) \}) \). Differently, \( P(a, \zeta_1, \zeta_2) \nott \subseteq P(a, \zeta_1, \zeta_1) \) and \( P(a, \zeta_1, \zeta_2) \nott \subseteq P(a, \zeta_1, \zeta_1) \). \( \square \)

The following Lemma follows straightforwardly from Definition 37.

**Lemma 39.** Let \( \Sigma \) be a set of WGTGDs and \( D \) an instance for a schema \( R \); let \( w \) be the maximum arity over all atoms in \( \Sigma \); also, let \( |R| \) be the number of predicate symbols in \( R \), and \( w \) be the maximum arity of a symbol in \( R \). The following claims hold.

1. For every atom \( a \in \text{chase}(\Sigma, D) \), we have \( |\text{cloud}(\Sigma, D, a)| \leq |R| \cdot (|\text{dom}(D)| + w)^w \); hence \( |\text{cloud}(\Sigma, D, a)| \) is polynomial in size in case the arity \( w \) is fixed, and exponential otherwise (assuming that \( |\text{dom}(D)| \geq 2 \)).

2. For each atom \( a \in \text{chase}(\Sigma, D) \), \( |\text{subclouds}(\Sigma, D, a)| \leq 2^{|R| \cdot (|\text{dom}(D)| + w)^w} \).

3. \( |\text{clouds}(\Sigma, D)/\Sigma| \leq 2^{|R| \cdot (|\text{dom}(D)| + w)^w} \), i.e., there are at most exponentially many possible clouds or subclouds in total in case the arity \( w \) is fixed. Similarly, we get:

4. \( |\text{clouds}^+(\Sigma, D)/\Sigma| \leq |\text{subclouds}^+(\Sigma, D)/\Sigma| \leq |R| \cdot (|\text{dom}(D)| + w)^w \cdot 2^{|R| \cdot (|\text{dom}(D)| + w)^w} \)

**Proof.**

1. All possible distinct atoms in a cloud are obtained by placing the symbols of \( a \), plus possibly symbols from \( \text{dom}(D) \), in at most \( w \) arguments of some predicate (relation) symbol in \( R \). The number of symbols to be placed is evidently \( \text{dom}(D) + w \), choosing among \( |R| \) relational predicates, hence the claim.

2. The number of ways we can choose \( \text{subclouds}(\Sigma, D, a) \) determines, as it is immediately seen, the set of all subsets of \( \text{cloud}(\Sigma, D, a) \), hence the claim.
(3) It is straightforwardly seen that the maximum set of all non-pairwise-isomorphic clouds in the chase has a number of elements that is bounded by the number of possible subclouds of a fixed atom; intuitively (and roughly), this holds because, since labeled nulls play the role of existentially quantified variables, what counts here is how the (at most \( w \)) nulls are placed in the atoms of the cloud together with the values of \( \text{dom}(D) \). From this, the claim immediately follows.

(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, their possible generating atom can have as arguments only among the \( \text{dom}(D) + w \) symbols with which we construct the subclouds.

\[
\square
\]

**Definition 40.** Let \( a \in \text{chase}(\Sigma, D) \). Then, we define the following:

- \( \downward_a \) denotes the set of all atoms that are nodes of the subtree of \( \text{GCF}(\Sigma, D) \) that is rooted in \( a \);
- \( \nabla_a = \downward_a \cup \text{cloud}(\Sigma, D, a) \);
- if \( S \) is a set of atoms in \( \text{GCF}(\Sigma, D) \), then \( \text{GCF}[a, S] \) is inductively defined as follows: (i) \( S \cup \{ a \} \subseteq \text{GCF}[a, S] \); (ii) if \( b \in \downward_a \) and \( b \) is obtained via the chase rule applied to a TGD \( \Phi \rightarrow \Psi \) via some some homomorphism \( \theta \) such that \( \theta(\Psi) = b \) (we remind that we have one-atom heads in TGDs, w.l.o.g.) and \( \theta(\Phi) \subseteq \text{GCF}[a, S] \), then \( b \in \text{GCF}[a, S] \).

The central importance of clouds in the context of weakly guarded TGDs is that if \( a \) is an atom of a generalized chase tree \( \text{GCF}(\Sigma, D) \), then \( \nabla_a \) is determined by \( \text{cloud}(\Sigma, D, a) \).

**Theorem 41.** If \( D \) is an instance for a schema \( \mathcal{R} \), \( \Sigma \) a set of WGTGDs, and \( a \in \text{chase}(\Sigma, D) \), then \( \nabla_a = \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \).

**Proof.** From the definition of \( \nabla_a \) and of \( \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \), it follows immediately that \( \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \subseteq \nabla_a \). We thus show the converse inclusion \( \nabla_a \subseteq \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \). Define \( \text{level}_a(a) = 0 \) and for each fact in \( b \in \text{cloud}(\Sigma, D, a) \), \( \text{level}_a(b) = 0 \), while for every other atom \( c \in \downward_a \), \( \text{level}_a(c) \) is the distance from \( a \) to \( c \) in \( \text{GCF}(\Sigma, D) \). Let \( U := \text{chase}(\Sigma, D) \). For each set \( X \subseteq \text{dom}(U) \), let \( U|_X = \{ c \in U \mid \text{dom}(c) \subseteq X \} \). We first show the following facts in parallel by induction on \( \text{level}_a(b) \):

1. If \( b \in \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \), then \( U|_{\text{dom}(b)} \subseteq \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \).
2. If \( b \in \nabla_a \) then \( b \in \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \).

**Induction basis.** In case \( \text{level}_a(b) = 0 \), \( b = a \), hence \( U|_{\text{dom}(b)} = U|_{\text{dom}(a)} = \text{cloud}(\Sigma, D, a) \subseteq \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \), which proves (1). Moreover, \( b = a \in \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \), which proves (2).

**Induction step.** Assume that (1) and (2) are satisfied for all \( c \) such that \( \text{level}_a(c) \leq i \). Assume \( \text{level}_a(c) = i + 1 \). \( b \) is produced by a TGD whose guard \( g \) matches some atom \( b^- \) having level \( i \), which is, by the induction hypothesis, in \( \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \). The body atoms of such TGD then match atoms whose arguments in \( \text{dom}(b^-) \) and are thus also in \( \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \). From the definition of \( \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \), we immediately derive that (2) holds for \( b \). To show (1), consider atoms \( b' \in U|_{\text{dom}(b)} \). In case \( \text{dom}(b') \subseteq \text{dom}(b^-) \), we have \( U|_{\text{dom}(b')} \subseteq U|_{\text{dom}(b^-)} \subseteq \text{GCF}[a, \text{cloud}(\Sigma, D, a)] \). Otherwise, \( b' \) has as argument some new variable \( X \) that was introduced during the generation of \( b \). Given that \( \Sigma \) is weakly guarded,
and each variable is introduced only once in the chase, there must be a chain from \( b \) to \( b' \) in \( GCF(\Sigma, D) \). A simple, further induction argument on \( level_b(b') \) shows that all applications of TGDs in that chain must have been fired on elements of \( GCF[\{a, \text{cloud}(\Sigma, D, a)\}] \) only. Therefore, in particular, \( b' \in GCF[\{a, \text{cloud}(\Sigma, D, a)\}] \). This proves (1). \qed

From the above theorem, we easily obtain the following result.

**Corollary 42.** If \( D \) is an instance for a schema \( \mathcal{R} \), \( \Sigma \) a set of WGTGDs, \( a,b \in \text{chase}(\Sigma, D) \), and \( (a, \text{cloud}(\Sigma, D, a)) \simeq (b, \text{cloud}(\Sigma, D, b)) \), then \( \nabla_a \simeq \nabla_b \).

**Definition 43.** Let \( a \) be an atom. The canonical renaming \( \text{can}_a : \text{dom}(a) \cup \text{dom}(D) \to \Delta_a \cup \text{dom}(D) \), where \( \Delta_a \) is an ad-hoc set \( \{\xi_1,\ldots,\xi_h\} \) of labelled nulls, not appearing in \( a \), is a substitution that maps each element of \( \text{dom}(D) \) into itself and maps the \( i \)-th argument value in lexicographic order of \( a \) which is not in \( \text{dom}(D) \) to \( \xi_i \), for all \( i \) such that \( 1 \leq i \leq h \), where \( h \) is the number of values in \( a \) that are not in \( \text{dom}(D) \). If \( S \subseteq \text{cloud}(\Sigma, D, a) \) (i.e., \( S \) is a subcloud of \( \text{cloud}(\Sigma, D, a) \)), then \( \text{can}_a(S) \) is well-defined and we denote by \( \text{can}(a, S) \) the pair \( (\text{can}_a(a), \text{can}_a(S)) \).

**Example 5.** If \( a = G(d,\zeta_1,\zeta_2,\zeta_1) \) where \( d \in \text{dom}(D) \) and \( \zeta_1,\zeta_2 \notin \text{dom}(D) \), and if \( S = \{P(\zeta_1), R(\zeta_2,\zeta_2), S(\zeta_1,\zeta_1,\zeta_1)\} \), then \( \text{can}_a(S) = G(d,\zeta_1,\zeta_2,\zeta_1) \), and \( \text{can}_a(S) = \{P(\zeta_1), R(\zeta_2,\zeta_1), S(\zeta_1,\zeta_1,\zeta_1)\} \). \qed

**Definition 44.** If \( D \) is an instance for a schema \( \mathcal{R} \), \( \Sigma \) set of WGTGDs on \( \mathcal{R} \), \( S \) is a set of atoms and \( a \in S \), then we write \( (\Sigma, a, S) \models Q \) iff there is a subtree \( T_a \) of \( GCF(\Sigma, S) \) that is rooted at \( a \) and a homomorphism (substitution) \( \theta \) which is the identity on \( D \), such that \( \theta(Q) \subseteq (S \cup \text{atoms}(T_a)) \), where \( \text{atoms}(T_a) \) are the vertices, i.e., atoms of \( T_a \).

The following result straightforwardly follows from from Theorem 41, from our previous definitions, and from a few additional considerations.

**Corollary 45.** If \( D \) is an instance for a schema \( \mathcal{R} \), \( \Sigma \) a set of WGTGDs, \( a \in \text{chase}(\Sigma, D) \), and \( Q \) is a conjunctive query, then \( \nabla_a \models Q \) iff \( (\Sigma, a, \text{cloud}(\Sigma, D, a)) \models Q \) iff \( (\Sigma, \text{can}_a(a), \text{can}_a(\text{cloud}(\Sigma, D, a))) \models Q \) iff there exists a subset \( S' \subseteq \text{cloud}(\Sigma, D, a) \) such that \( (\Sigma, \text{can}_a(a), \text{can}_a(S')) \models Q \).

Towards the aim of designing an alternating algorithm that computes the relevant parts of \( \text{chase}(\Sigma, D) \) necessary to answer a query \( Q \), we can thus reduct any subcomputation of the form \( (a, S) \) where \( S \subseteq \text{cloud}(\Sigma, D, a) \) to \( \text{can}(a, S) \). Note also that each pair \( \text{can}(a, \text{cloud}(\Sigma, D, a)) \) can be seen as the unique canonical representative of the equivalence class \( \{b, \text{cloud}(\Sigma, D, b)\} \) for \( (\Sigma, \text{cloud}(\Sigma, D, a)) \). The algorithm uses as basic data structures (configurations) tuples of the form \( (a, S, S^+, \prec, b) \). Each such tuple corresponds to a vertex of the chase-tree at some moment in time. The informal meaning of the parameters is as follows:

23
(1) \(a\) is the atom of the chase tree under consideration;

(2) \(S\) is the set \(\text{cloud}(\Sigma, D, a)\) or a subset thereof;

(3) \(S^+\) is the subset of \(\text{cloud}(\Sigma, D, a)\) that has been established so far;

(4) \(<\) is a total ordering of the atoms in \(S\) that corresponds to the temporal ordering in which the atoms of \(S\) are actually derived in \(\text{chase}(\Sigma, D)\);

(5) \(b\) is an atom that needs to be derived. In some special cases (namely, on the “main” path in the proof tree developed by \(A\text{check}\)), the algorithm will not try to derive a specific atom, but will just try to match the input query atom \(P(t_1, \ldots, t_r)\) against the atoms of that path; in that case, we shall use the symbol \(\ast\) in place of \(b\).

We are now ready for describing (at a somehow high level) how the algorithm \(A\text{check}\) algorithm works. The algorithm first checks whether \(D = Q\) already. If so, \(A\text{check}\) returns “true” and stops. Otherwise, the algorithm attempts to guess a path that contains an atom \(q'\) that is an instance of \(Q\).

We first explain the initialization. The algorithm \(A\text{check}\) starts at \(D\) and guesses some atom \(a\) of \(D\), to be expanded into a main branch that will eventually lead to an atom \(q'\) matching \(Q\). To this aim, the algorithm also guesses a set \(S \subseteq \text{cloud}(\Sigma, D, a)\) and a total order \(<\) on \(S\), and generates a configuration \((a, S, S^+, <, \ast)\).

Assume the set \(S\) is given by \(S_D \cup S^+\), where \(S^+ = \{a_1, a_2, \ldots, a_k\}\), where \(S_D \subseteq D\), and for each \(1 \leq i \leq k\), \(a_i \notin D\). The total order \(<\) is such that all elements of \(S_D\) precede those of \(S^+\). Assume that \(<\) on \(S^+\) is: \(a_1 < a_2 < \cdots < a < \cdots < a_k\). To prove that \(S\) is actually a subset of \(\text{cloud}(\Sigma, D, a)\), it is necessary for \(A\text{check}\) to prove that each of the atoms \(a_1, \ldots, a_k\) is indeed an atom of \(\text{chase}(\Sigma, D)\), where the proof for each atom \(a_i\) may assume only \(<\) smaller atoms of \(S\) as premises. The algorithm thus finds atoms \(a_1, \ldots, a_k\) in \(D\) to be expanded to proof trees for \(a_1, \ldots, a_k\), respectively. For each \(1 \leq i \leq k\), it generates a configuration \((d_i, S, S_D \cup \{a_1, a_2, \ldots, a_{i-1}\}, a_i)\). Informally, each one of such configurations requires to prove \(a_i\) assuming that \(a_1, \ldots, a_{i-1}\) have all been proved. \(A\text{check}\) thus simulates the sequential proof of all atoms in \(\text{cloud}(\Sigma, D, a)\) of the original chase via a parallel universal branching.

We now explain how the configuration tree is expanded at each configuration \(c\). Let \(c = (a, S, S', <, b)\), where \(S = \{a_1, a_2, \ldots, a_k\}\), \(S' = \{a_1, a_2, \ldots, a_i\}\), and \(<\) is given by \(a_1 < a_2 < \cdots < a < \cdots < a_k\). If \(b \in D\), then \(A\text{check}\) accepts this configuration, and does not further expand it. If \(b = \ast\), then \(A\text{check}\) checks (via a simple existential subroutine) whether \(Q\) matches \(a\), i.e., if \(a\) is the homomorphic image of the input query atom \(P(t_1, \ldots, t_r)\). If so, \(A\text{check}\) accepts \(c\) and does not further expand it. In case \(b \neq \ast\), \(A\text{check}\) checks whether \(b = a\). If so, \(A\text{check}\) accepts the configuration \(c\) and does not further expand it. Otherwise, the configuration tree is expanded as follows. \(A\text{check}\) guesses a TGD \(\rho \in \Sigma\), where \(\rho = \Phi \rightarrow \Psi\), whose guard \(g\) matches \(a\) via some substitution \(\theta\) and such that: \(\theta(\Phi) \subseteq S'\), and \(\theta(\Psi)\) is the new atom generated (possibly containing some fresh labeled nulls in \(\Delta_N\)). Before creating the actual new configuration \(c_1\) from \(c\), let us – for the sake of better intelligibility – present a preliminary new configuration \(c^*\). We have \(c^* = (a_1, S_1, S_1', <_1, b)\), where:

(a) \(a_1 = \theta(\Psi)\) is the new atom generated by the application of \(\rho\) with substitution \(\theta\) (we recall that all our TGDs have a single atom in their head).

(b) \(S_1\) contains \(a_1\) and each atom \(D\) of \(S\) such that \(\text{dom}(d) \subseteq \text{dom}(a_1)\). Thus, in addition to the new atom \(a_1\), \(S_1\) inherits all atoms that were in the subcloud \(S\) of the parent configuration \(c\), and that, moreover, are compatible with the vocabulary of \(a_1\). In addition, \(S_1\), which intuitively represents the cloud or a subcloud of \(a_1\), may contain a set \(\text{newatoms}(c^*)\)
of further atoms that are guessed by the Acheck algorithm, and which must each contain at least one fresh variable (labeled null) of $a_1$, or one domain element $c \in D$ that does not occur in $S$.

(c) $S'_1$ is such that $S'_1 := S_1$. Intuitively, $c^*$ represents the “main” descendant of $c$, where we assume that already all atoms of the guessed subcloud $S$ have been proved. As described later on, Acheck will have to generate in parallel further configurations, that actually prove the atoms of the set $\text{newatoms}(c^*)$.

(d) $\prec_1$ is a total order that is obtained from $\prec$ by eliminating all atoms $e$ such that $\text{dom}(e) \not\subseteq \text{dom}(e_1)$, and by inserting the atoms of $\text{newatoms}(c^*)$ at nondeterministically chosen places, but after all atoms from the set $\text{oldproved}(c^*)$, which consists of those atoms of $S_1$ that also occur in $S'$ (and that were thus already assumed to be proved at the parent configuration $c$).

(e) $b_1$ is such that $b_1 := b$.

Importantly, instead of generating the above-described configuration $c^* = (a_1, S_1, S'_1, \prec_1, b_1)$, the Acheck algorithm actually generates the following configuration $c'$: $c' = \text{can}_{a_1}(c_1) = (\text{can}_{a_1}(a_1), \text{can}_{a_1}(S_1), \text{can}_{a_1}(S'_1), \text{can}_{a_1}(\prec_1), \text{can}_{a_1}(b_1))$, where $\text{can}_{a_1}(\prec_1)$ is the total order on the atoms of $\text{can}_{a_1}(S'_1)$, derived from $\prec_1$. The algorithm will furthermore generate in parallel, and in a universal computation, a set of auxiliary descendant-configurations of $c$ for proving that the guessed atoms in $\text{can}_{a_1}(\text{newatoms}(c^*))$ are actually derivable. Let the linear order $\prec_1$ of the set $S_1$ of $c^*$ be a concatenation of the order $\prec$, restricted to $\text{oldproved}$, and the ordered list $H_1 \prec H_2 \prec \cdots \prec H_k$. For each $1 \leq i \leq k$, Acheck generates a configuration $c'_i$ of the following form: $c'_i = \text{can}_{a_1}(c^*_i)$, where $c^*_i = (a_1, S_1, \text{oldproved} \cup \{H_1, \ldots, H_{i-1}\}, \prec_1, H_i)$. This completes the description of the Acheck algorithm.

**Theorem 46.** The Acheck algorithm is correct and works in exponential time in case of bounded arities, and in doubly exponential time otherwise.

**Proof.** It is easy to see that the algorithm is sound with respect to the standard chase, i.e., if $\text{Acheck}(\Sigma, D, Q)$ returns “true”, then $\text{chase}(\Sigma, D) \models Q$. In fact, the algorithm performs, modulo variable renamings which preserve soundness according to Corollary 45, essentially nothing but chase steps starting from $D$ and $\Sigma$, even though not necessarily in the same order as 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 satisfied by some chase, and thus also by the standard chase $\text{chase}(\Sigma, D)$.

The completeness of Acheck with respect to $\text{chase}(\Sigma, D)$ can be seen as follows. Whenever $\text{chase}(\Sigma, D) \models Q$, there is a finite proof of $Q$, i.e., a finite sequence $\xi$ of generated atoms that ends with some atom $q'$ which is an instance of $Q$. This proof can be simulated by the alternating computation Acheck by using the following guidelines: (i) steer the main branch of Acheck towards (a variant of) $q'$ by choosing successively the same TGDs and substitutions $\theta$ (modulo the appropriate variable renamings) as those used in the standard chase for the branch of $q'$; (ii) whenever a subcloud $S$ has to be chosen for some atom $a$ by Acheck, choose the set of atoms $\text{cloud}(\Sigma, D, a) \cap (D \cup \{\text{atoms}(\xi)\})$, modulo appropriate variable renaming; (iii) for the ordering $\prec$, always choose the one given by $\xi$. The fact that no $Q$-instance is lost when replacing configurations $c^*$ by their canonical versions $c' = \text{can}_{a_1}(c^*)$ is guaranteed by Corollary 45.

In case of a fixed arity, the size of each configuration $c$ is polynomial in $\Sigma \cup D$. Thus, Acheck describes an alternating PSPACE (i.e., APSPACE) computation. It is well-known that $\text{APSPACE} =
Proof. It is sufficient to start with an empty set \( X \) and then cycle over each possible ground atom \( q \) of the Herbrand base \( HB(D) \), and check whether \( \text{chase}(\Sigma, D) \models q \), and if so, add it to \( X \). The result is \( \text{chase}^+(\Sigma, D) \). The claimed time bounds follow straightforwardly. \( \square \)

We now show that answering general conjunctive queries is of the same complexity. To this aim, we will use squid decompositions as defined in Section 3.

**Theorem 48.** Let \( \Sigma \) be a set of WGTGDs, \( D \) an instance for a schema \( R \), and \( Q \) a Boolean conjunctive query. Determining whether \( DU \Sigma \models Q \), or, equivalently, whether \( \text{chase}(\Sigma, D) \models Q \), is feasible in \text{EXPTIME} in case of bounded arities, and in \text{2-EXPTIME} in general.

**Proof.** We construct a nondeterministic algorithm \( \text{Qcheck} \) such that \( \text{Qcheck}(\Sigma, D, Q) \) outputs “true” iff \( D \cup \Sigma \models Q \), or, equivalently, iff \( \text{chase}(\Sigma, D) \models Q \). The algorithm heavily relies on the concept of squid decompositions, and on Lemma 36. \( \text{Qcheck} \) works as follows.

1. \( \text{Qcheck} \) computes \( \text{chase}^+(\Sigma, D) \).
2. \( \text{Qcheck} \) nondeterministically guesses a squid decomposition \( \delta = (Q^+, h, H, T) \) of \( Q \) based on a set \( V_\delta \subseteq \text{var}(\langle h(Q^+) \rangle) \), where \( H = \{ g \in h(Q^+) \mid \text{var}(g) \subseteq V_\delta \} \), where \( T \) is \( \{ V_\delta \} \)-acyclic, and \( \text{Qcheck} \) also guesses a substitution \( \theta_0 : V_\delta \rightarrow \text{dom}(D) \) such that \( \theta_0(H) \subseteq \text{chase}^+(\Sigma, D) \). Note that this is an NP-guess, because the size of \( Q^+ \) is at most twice the size of \( Q \).
3. \( \text{Qcheck} \) tests whether \( \theta_0 \) can be extended to a homomorphism \( \theta \) such that \( \theta(T) \subseteq \text{chase}^+(\Sigma, D) \). Note that by Lemma 36, this is equivalent to \( \text{chase}(\Sigma, D) \models Q \). Such a \( \theta \) exists iff for each connected subgraph \( t \) of \( \theta_0(T) \), there is a homomorphism \( \theta_t \) that leaves all elements of \( \text{dom}(D) \) invariant such that \( \theta_t(t) \subseteq \text{chase}^+(\Sigma, D) \). The \( \text{Qcheck} \) algorithm thus identifies the connected components of \( \theta_0(T) \). Each such component is a \( \{ \text{dom}(D) \} \)-acyclic conjunctive query, some of whose arguments may contain constants from \( \text{dom}(D) \). Each such component can thus be represented in form of a \( \{ \text{dom}(D) \} \)-join tree \( t \). For each such join tree \( t \), \( \text{Qcheck} \) now tests whether there exists a homomorphism \( \theta_t \) (that, we remind, restricted to \( D \) is the identity) such that \( \theta_t(t) \subseteq \text{chase}^+(\Sigma, D) \). This is done by the subroutine \( \text{Tcheck} \), that takes as arguments the TGDs, the database instance, and the subgraph \( t \) of \( \theta_0(T) \); how \( \text{Tcheck}(\Sigma, D, t) \) is executed is described below.

4. \( \text{Qcheck} \) outputs “true” iff all steps (1), (2) and (3) have a positive result.

The correctness of \( \text{Qcheck} \) follows from Lemma 36. Given that step (2) is nondeterministic, the complexity of \( \text{Qcheck} \) is in \text{NP}^X \( \text{, i.e., NP with an oracle in } X \), where \( X \) is a complexity class that is sufficiently powerful for: (i) computing \( \text{chase}^+(\Sigma, D) \), and (ii) performing the tests \( \text{Tcheck}(\Sigma, D, t) \).

We now describe the \( \text{Tcheck} \) subroutine. \( \text{Tcheck}(\Sigma, D, t) \) can be obtained from \( \text{Acheck} \) via the following modifications. In addition to the data structures carried by each configuration of \( \text{Acheck} \), each configuration of \( \text{Tcheck} \) also maintains an array \( \text{subst} \) of length \( w \), where \( w \) is the maximum arity of all predicates in \( R \) (\( R \), we remind the reader, is the relational schema we are considering). Each array element of \( \text{subst} \) describes a substitution that replaces some element \( x \in \text{dom}(t) - \text{dom}(D) \) of \( t \) by some element from \( \{ x_1, x_2, \ldots \} \), where the \( x_i \) are the new “canonical” elements dynamically generated by \( \text{Tcheck} \) (see the description of \( \text{Acheck} \), where the
generation of the canonical elements is the done in the same way). Moreover, each configuration of Tcheck maintains a pointer Tpoint to a vertex (i.e., atom) of t, which informally points to the root of the subtree of t that still needs to be matched by descendent configurations of c.

Tcheck works like Acheck, but instead of nondeterministically constructing a main configuration path of the configuration tree such that eventually some atom matches the unique query atom, Tcheck nondeterministically constructs a main configuration (sub)tree τ of the configuration tree, such that eventually all atoms of the join tree t will be consistently translated into some vertices of τ. An important component of each main configuration c of Tcheck is its current atom a. Initially, the atom a is some nondeterministically chosen atom of D. For deeper main configurations of the alternating computation tree, a will take on values of nodes of \( GCF(\Sigma, D) \).

The computation, similarly to Acheck, starts by generating initially a configuration \( (a, S, S_+, \prec, \ast, \text{Tpoint}, \text{subst}) \), where a is, as said, nondeterministically chosen from the database D, and where Tpoint points to the root r of t, and subst encodes a homomorphism σ such that \( \sigma(r) = a \), if such a homomorphism exists, and subst is empty otherwise. This configuration will now be the root of the main configuration tree. In addition, just as in Acheck, Tcheck generates further configurations whose task is to prove that all elements of S are indeed provable.

In general, the pointer π of each main configuration c points to some atom \( \pi^1 \) 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 \( GCF(\Sigma, D) \) that satisfies the subquery of t rooted at \( \pi^1 \). More precisely, the expansion of a main configuration \( c = (a, S, S_+, \prec, \ast, \text{Tpoint}, \text{subst}) \) works as follows. For a configuration c, Tcheck first checks whether there exists a homomorphism \( \sigma : \text{dom}(\text{subst}(a_q)) \rightarrow \text{dom}(a) \) such that \( \sigma(\text{subst}(a_q)) = a \). If so, and if \( \pi^1 \) is a leaf of t, then the current configuration turns into an accepting one.

If a suitable homomorphism σ exists and if \( \pi^1 \) is not a leaf of t, then Tcheck nondeterministically decides whether σ 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 \( GCF(\Sigma, D) \). In case of a good match, Tcheck nondeterministically generates for each child \( a_q' \) of \( a_q \) in t a new configuration \( c' = \text{can}_{a}(a_q', S', S_+, \prec', \ast, \text{Tpoint}', \text{subst}') \), where Tpoint' points to \( a_q' \), and where subst' encodes can_{a}(σ). Otherwise (in case of no match or no good match), the configuration is rejecting in case of a leaf of t, and in case of an inner node of t, a child configuration \( c' = (a_q', S, S_+, \prec, \ast, \text{Tpoint}, \text{subst}) \) of c is nondeterministically created, whose first component is a child \( a_q' \) of \( a_q \), and where \( c' \) inherits its other components from c. Of course, Tcheck, just as Acheck, generates, in auxiliary configurations in order to prove that all atoms of S are actually derivable.

The correctness of Tcheck can be shown along similar lines as the one of 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 follows, as said from the correctness of Tcheck and from Lemma 36.

As for the complexity of Qcheck, note that in case the arity is bounded, Tcheck runs in \( \text{APSPACE} = \text{EXPTIME} \), and computing \( \text{chase}^+(\Sigma, D) \) is in \( \text{EXPTIME} \) by Corollary 47. Thus, Qcheck runs in time \( \text{NP}^{\text{EXPTIME}} = \text{EXPTIME} \). In case of unbounded arities, both computing \( \text{chase}^+(\Sigma, D) \) and running Tcheck are in 2-\( \text{EXPTIME} \), therefore Qcheck runs in time \( \text{NP}^{2-\text{EXPTIME}} = 2-\text{EXPTIME} \).

By combining Theorem 30 and Theorem 48 we immediately get the following complexity characterization for reasoning under WGTGDs.

**Theorem 49.** Let \( \Sigma \) be a set of WGTGDs, let D be an instance, and let Q be a Boolean conjunctive query. Determining whether \( D \cup \Sigma \models Q \), or, equivalently, whether \( \text{chase}(\Sigma, D) \models Q \)
is \text{EXPTIME} complete in case of bounded predicate arities, and even in case \Sigma is fixed; it is 2-\text{EXPTIME} complete in general. The same completeness results hold for the problem of query containment under WGTGDs.

### 4.4 Guarded TGDs

Let us now turn our attention to GTGDs.

**Theorem 50.** Let \Sigma be a set of GTGDs over a schema \mathcal{R}, and let \mathcal{D} be an instance for \mathcal{R}. Let, moreover \( w \) denote the maximum arity of any predicate appearing in \mathcal{R}, and let \(|\mathcal{R}|\) denote the total number of predicate symbols. Then:

1. Computing \( \text{chase}^{-1}(\Sigma, \mathcal{D}) \) can be done in polynomial time if both \( w \) and \(|\mathcal{R}|\) are bounded, and thus also in case of a fixed set \Sigma. This problem is in exponential time in case \( w \) is bounded, and in double-exponential time otherwise.

2. If \( Q \) is an atomic query, then checking whether \( \text{chase}(\Sigma, \mathcal{D}) \models Q \) is \text{PTIME}-complete in case both \( w \) and \(|\mathcal{R}|\) are bounded, and remains \text{PTIME}-complete even in case \Sigma is fixed. This problem is \text{EXPTIME}-complete if \( w \) is bounded and 2-\text{EXPTIME}-complete in general. It remains 2-\text{EXPTIME}-complete even when \(|\mathcal{R}|\) is bounded.

3. If \( Q \) is a general conjunctive query, checking whether \( \text{chase}(\Sigma, \mathcal{D}) \models Q \) is \text{NP}-complete in case both \( w \) and \(|\mathcal{R}|\) are bounded, and thus also in case of a fixed set \Sigma. Checking whether \( \text{chase}(\Sigma, \mathcal{D}) \models Q \) is \text{EXPTIME}-complete if \( w \) is bounded and 2-\text{EXPTIME}-complete in general. It remains 2-\text{EXPTIME}-complete even when \(|\mathcal{R}|\) is bounded.

4. Query containment under GTGDs is \text{NP}-complete if both \( w \) and \(|\mathcal{R}|\) are bounded, and even in case the set \Sigma of GTGDs is fixed.

5. Query containment under GTGDs is \text{EXPTIME}-complete if \( w \) is bounded and 2-\text{EXPTIME}-complete in general. It remains 2-\text{EXPTIME}-complete even when \(|\mathcal{R}|\) is bounded.

**Proof.** The \text{PTIME}-hardness of checking \( \text{chase}(\Sigma, \mathcal{D}) \models Q \) for atomic queries \( Q \) and for fixed \Sigma follows from the fact that factual inference in fully guarded Datalog programs is \text{PTIME}-hard. In fact, in the proof of Theorem 4.4 of [11] it is shown that fact inference from a single-rule Datalog program whose body contains a guard atom that contains all variables is \text{PTIME}-hard.

\text{NP}-hardness in items (3) and (4) is immediately derived from hardness of containment (which in turn is polynomially equivalent to query answering) without constraints [9].

The hardness results for \text{EXPTIME} and 2-\text{EXPTIME} are all derived by minor variants of Theorem 30. However, in case \(|\mathcal{R}|\) is unbounded and \( w \) is bounded, the tape cells of the polynomial worktape will be simulated by using polynomially many predicate symbols. For example, the fact that in configuration \( v \) cell 5 contains a 1 may be encoded as \( S^5_3(v) \), and so on.

The membership results are proved exactly as those for WGTGDs, except that instead of using the concept of cloud, we use the similar concept of \text{restricted cloud}. The restricted cloud \( r\text{cloud}(\Sigma, \mathcal{D}, a) \) of an atom \( a \in \text{chase}(\Sigma, \mathcal{D}) \) is the set of all atoms \( b \in \text{chase}(\Sigma, \mathcal{D}) \) such that \( \text{dom}(b) \subseteq \text{dom}(a) \). By a proof that is almost identical to the one of Theorem 41, we can show that if \( D \) is an instance, \( \Sigma \) a set of GTDs, and if \( a \in \text{chase}(\Sigma, \mathcal{D}) \), then \( \nabla^r(a) = \text{GCF}[a, r\text{cloud}(\Sigma, \mathcal{D}, a)] \), where \( \nabla^r(a) = \{v \} \cup r\text{cloud}(\Sigma, D, a) \). It follows that, for the main computational tasks, we can use algorithms \( r\text{Acheck}, r\text{Qcheck} \), and \( r\text{Tcheck} \) that differ from the respective original algorithms only in that \( r\text{cloud}(\cdot, \cdot, \cdot) \) instead of \( \text{cloud}(\cdot, \cdot, \cdot) \) is used. However, while in case both \(|\mathcal{R}|\) and \( w \) are bounded, a cloud or subcloud can still be of polynomial size in \(|D \cup \Sigma|\), a restricted cloud \( r\text{cloud}(\Sigma, \mathcal{D}, a) \) has only constantly many atoms, and the memorization of its canonical version \( \text{can}_{a}(r\text{cloud}(\Sigma, \mathcal{D}, a)) \) 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 \( r \)Acheck and of \( rTcheck \) only require logarithmic memory space. Since \( a \)logspace = \( \text{PTIME} \), the results for items (1) and (2) for the case both \( w \) and \( |R| \) are bounded follow. The \( rQcheck \) algorithm then runs in \( \text{NP}^{\text{PTIME}} = \text{NP} \), and hence also our deciding \( \text{chase}(\Sigma, D) \models Q \) is in \( \text{NP} \). Item (3) follows immediately from Item (2) and Corollary 8. \( \square \)

Note that one of the main results of Johnson and Klug [18], namely, that query containment under inclusion dependencies of bounded arities is \( \text{NP} \)-complete, is a special case of Item (3) of Theorem 50.

5 Application

In this section we show the application of our subset of Datalog\(^3\) to a formalism called F-logic Lite; we show that query answering and containment under F-logic Lite rules are \( \text{NP} \)-complete.

F-logic Lite is a smaller but expressive version of F-logic, a well-known formalism introduced for object-oriented deductive databases. We refer the reader to refer the reader to [4] for details about F-logic Lite. Roughly, with respect to F-Logic, F-logic Lite excludes negation and default inheritance, and allows only a limited form of cardinality constraints.

We now encode F-logic Lite using Datalog\(^3\) rules, that we denote with \( \Sigma_{FLL} \), with \( \Sigma_{FLL} = \{ \rho_i \}_{1 \leq i \leq 12} \).

\[
\begin{align*}
(1) & \quad \text{member}(V, T) \leftarrow \text{type}(O, A, T), \text{data}(O, A, V). \\
(2) & \quad \text{sub}(C_1, C_2) \leftarrow \text{sub}(C_1, C_3), \text{sub}(C_3, C_2). \\
(3) & \quad \text{member}(O, C) \leftarrow \text{member}(O, C), \text{sub}(C, C_1). \\
(4) & \quad V = W \leftarrow \text{data}(O, A, V), \text{data}(O, A, W), \text{funct}(A, O). \\
& \quad \text{Note that this is the only EGD in this axiomatization.} \\
(5) & \quad \text{data}(O, A, V) \leftarrow \text{mandatory}(A, O). \\
& \quad \text{Note that this is a TGD with an existential variable in the head (variable } V; \text{ quantifiers are omitted).} \\
(6) & \quad \text{type}(O, A, T) \leftarrow \text{member}(O, C), \text{type}(C, A, T). \\
(7) & \quad \text{type}(C, A, T) \leftarrow \text{sub}(C, C_1), \text{type}(C_1, A, T). \\
(8) & \quad \text{type}(C, A, T) \leftarrow \text{type}(C, A, T_1), \text{sub}(T_1, T). \\
(9) & \quad \text{mandatory}(A, C) \leftarrow \text{sub}(C, C_1), \text{mandatory}(A, C_1). \\
(10) & \quad \text{mandatory}(A, O) \leftarrow \text{member}(O, C), \text{mandatory}(A, C). \\
(11) & \quad \text{funct}(A, C) \leftarrow \text{sub}(C, C_1), \text{funct}(A, C_1). \\
(12) & \quad \text{funct}(A, O) \leftarrow \text{member}(O, C), \text{funct}(A, C).
\end{align*}
\]

It can be easily shown that the only EGD in the above Datalog\(^3\) rules does not actually interact with the TGDs, and therefore we can ignore it.

We now prove the complexity results.

**Theorem 51.** Conjunctive query answering under F-logic Lite rules is \( \text{NP} \)-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 \( \text{data}(X, V_i, V_j) \) and \( \text{data}(X, V_j, V_i) \), where \( X \) is a unique, fixed variable. Let \( D \) be the instance \( D = \{ \text{data}(o_i, r_i, q_i), \text{data}(o_i, g_i, r_i), \text{data}(o_i, r_i, b_i), \text{data}(o_i, b_i, r_i), \text{data}(o_i, g_i, b_i), \text{data}(o_i, b_i, g_i) \} \). 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. This proves the claim. \( \square \)

**Theorem 52.** Conjunctive query answering under F-logic Lite rules is in \( \text{NP} \).
PROOF (sketch). As mentioned before, we can ignore the only EGD in $\Sigma_{FLL}$, since it does not interfere with query answering. Let us denote with $\Sigma'_{FLL}$ the set of Datalog\textsuperscript{2} resulting from $\Sigma_{FLL}$ by eliminating rule $\rho_1$, i.e., let $\Sigma'_{FLL} = \Sigma_{FLL} - \{\rho_1\}$. To establish membership in NP, it is sufficient to show that:

1. $\Sigma'_{FLL}$ is weakly guarded.
2. $\Sigma'_{FLL}$ is such that, for every instance $D$, there are up to $D$-isomorphisms, polynomially many clouds; more precisely, for every instance $D$ there exists a polynomial $\text{pol}$ such that $|\text{clouds}(\Sigma, D)| \leq \text{pol}(|D|)$.
3. There is a polynomial $\text{pol}'(\cdot)$ such that for each instance $D$ and for each atom $a$: (3.1) if $a \in D$, then $\text{cloud}(\Sigma, D, a)$ can be computed in time $\text{pol}'(|D|)$, and (3.2) if $a \notin D$, then $\text{cloud}(\Sigma, D, a)$ can be computed in time $\text{pol}(|D|)$ from $D$, $a$, and $\text{cloud}(\Sigma, D, b)$, where $b$ is the predecessor of $a$ in GCF($\Sigma, D$).

Under the above condition, the membership in NP can be proved by exhibiting the following.

(i) An algorithm, analogous to $\text{Acheck}$, that constructs all “canonical” versions of the atoms of the chase and their clouds (the latter are stored in a “cloud store”), in polynomial time; then, checks whether a given (Boolean) query is satisfied by some atom in the cloud store.

(ii) An algorithm, analogous to $\text{Qcheck}$, that guesses (by calling an analogous version of $\text{Tcheck}$) entire clouds by guessing their index (a unique integer) in the cloud store, and checks in alternating logarithmic space (algspace) the correctness of the cloud guess, using in addition only the cloud of the main atom of the predecessor configuration. The complexity of running this algorithm is shown to be NPalgspace = NP.

(1) is readily seen: the affected positions are the following: $\text{data}[3], \text{member}[1], \text{type}[1], \text{mandatory}[2], \text{func}[2], \text{data}[1]$. It is easy to see that every rule of $\Sigma'_{FLL}$ is weakly guarded, and thus $\Sigma_{FLL}$ is weakly guarded.

Now let us sketch (2). Let $\Sigma^\text{full}_{FLL} = \Sigma_{FLL} - \{\rho_5\}$, i.e., the set of all TGDs of $\Sigma_{FLL}$ but $\rho_5$. These are all full TGDs and their application does not alter the domain. We have $\text{chase}(\Sigma^\text{full}_{FLL}, D) = \text{chase}(\Sigma'_{FLL}, \text{chase}(\Sigma^\text{full}_{FLL}, D))$. Let us now have a closer look at $D^+ = \text{chase}(\Sigma^\text{full}_{FLL}, D)$. Clearly, $\text{dom}(D^+) = \text{dom}(D)$. For each predicate symbol $P$, let $\text{Rel}(P)$ denote the relation consisting of all $P$-tuples in $D^+$. Let $\Omega$ be the family of all relations that can be obtained from any of the relations $\text{Rel}(P)$ by performing an arbitrary selection followed by some projection (we forbid disjunctions in the selection predicate). For example, assume $c, d, e \in \text{dom}(D)$; then, $\text{Rel}([\text{data}])$ will give rise to relations $\pi_1, \pi_2, \pi_3$ of $\text{data}$, and so on, where the numbers are attribute identifiers (the notation here should be self-explanatory). Given that $D^+$ is of size polynomial in $D$ and that the maximum arity of any relation $\text{Rel}(P)$ is 3, the set $\Omega$ is of size polynomial in $D^+$ and thus polynomial in $D$. It can now be shown that $\Omega$ is preserved in a precise sense, when going to the final result $\text{chase}(\Sigma'_{FLL}, D^+)$. In particular, for each relation $\text{Rel}(P)$ corresponding to predicate $P$ in the final chase result, when performing a selection on $\text{Rel}(P)$ that assigns fixed values not in $\text{dom}(D)$ to one or more attributes, and projecting on the other columns, the set of all tuples of $\text{dom}(D)$-elements in the result is a relation in $\Omega$. For example, assume that $v_5$ is a specific labeled null, then the set of all $T \in \text{dom}(D)$ such that $\text{member}(v_5, T)$ is an element of the final result is a set in $\Omega$; similarly, if $v_7$ and $v_8$ are new values, the set of all values $A$ such that $\text{data}(v_7, A, v_8)$ is a relation in $\Omega$. It is easy to see that from this it follows that $\Sigma_{FLL}$ satisfies (2).

In fact, all possible clouds are determined by the polynomially many ways of choosing at most three elements of $\Omega$ 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 $\rho_5$, we just analyze which sets of values (or tuples) are attached to it via rules $\rho_1$, then $\rho_6$, $\rho_7$, $\rho_8$, $\rho_{10}$, and so on, and conclude that these sets were all already present at the next lower
level, and thus, by induction hypothesis, are in $\Omega$. Condition (3) can straightforwardly proved by similar arguments.

From Theorems 51 and 52 we immediately get:

**Corollary 53.** Conjunctive query answering under F-logic Lite rules is NP-complete.

6 Conclusions

In this paper we identified a large and non-trivial class of tuple- and equality-generating dependencies for which the problems of containment and answering for conjunctive queries are decidable, and provided the relevant complexity results. Applications of this result include databases and knowledge representation. In particular, we have shown that this class of constraints subsumes the classical work of Johnson and Klug [18] as well as (with some extension not detailed in this paper) the more recent results from [4].

A related previous approach to guarded logic programming is guarded open answer set programming [17]. It is easy to see that a set of GTGDs can be interpreted as a guarded answer set program as defined in [17], but that guarded answer set programs are, in general, more expressive than GTGDs, for example, because they allow for negation. Investigating the decidability and complexity of query answering (and containment) under more expressive classes of constraints, capable of subsuming, for instance, the results of [3] and [17], is the subject of our future work. We also plan to investigate the same problem in the case of finite models.

Acknowledgments. Andrea Calì and Georg Gottlob were supported by the EPSRC project “Schema Mappings and Automated Services for Data Integration and Exchange” (EP/E010865/1). Georg Gottlob’s work was also supported by a Royal Society Wolfson Research Merit Award.

References


32