THE UNIVERSITY of EDINBURGH

# Edinburgh Research Explorer

# Semantic Acyclicity Under Constraints

OPEN ACCESS

# Semantic Acyclicity Under Constraints

Pablo Barceló
Center for Semantic Web
Research & DCC
University of Chile
pbarcelo@dcc.uchile.cl

Georg Gottlob
Department of Computer
Science
University of Oxford
georg.gottlob@cs.ox.ac.uk

Andreas Pieris
Institute of Information
Systems
TU Wien
pieris@dbai.tuwien.ac.at

## ABSTRACT

A conjunctive query (CQ) is semantically acyclic if it is equivalent to an acyclic one. Semantic acyclicity has been studied in the constraint-free case, and deciding whether a query enjoys this property is NP-complete. However, in case the database is subject to constraints such as tuple-generating dependencies (tgds) that can express, e.g., inclusion dependencies, or equality-generating dependencies (egds) that capture, e.g., functional dependencies, a CQ may turn out to be semantically acyclic under the constraints while not semantically acyclic in general. This opens avenues to new query optimization techniques. In this paper we initiate and develop the theory of semantic acyclicity under constraints. More precisely, we study the following natural problem: Given a CQ and a set of constraints, is the query semantically acyclic under the constraints, or, in other words, is the query equivalent to an acyclic one over all those databases that satisfy the set of constraints?

We show that, contrary to what one might expect, decidability of CQ containment is a necessary but not sufficient condition for the decidability of semantic acyclicity. In particular, we show that semantic acyclicity is undecidable in the presence of full tgds (i.e., Datalog rules). In view of this fact, we focus on the main classes of tgds for which CQ containment is decidable, and do not capture the class of full tgds, namely guarded, non-recursive and sticky tgds. For these classes we show that semantic acyclicity is decidable, and its complexity coincides with the complexity of CQ containment. In the case of egds, we show that if we focus on keys over unary and binary predicates, then semantic acyclicity is decidable (NP-complete). We finally consider the problem of evaluating a semantically acyclic query over a database that satisfies a set of constraints. For guarded tgds and functional dependencies the evaluation problem is tractable.

## 1. INTRODUCTION

Query optimization is a fundamental database task that amounts to transforming a query into one that is arguably more efficient to evaluate. The database theory community has developed several principled methods for optimization of conjunctive queries (CQs), many of which are based on *static-analysis* tasks such as containment [1]. In a nutshell, such methods compute a *minimal* equivalent version of a CQ, where minimality refers to number of atoms. As argued by Abiteboul, Hull, and Vianu [1], this provides a theoretical notion of "true optimality" for the reformulation of a CQ, as opposed to practical considerations based on heuristics. For each CQ $q$ the minimal equivalent CQ is its *core* $q'$ [21]. Although the static analysis tasks that support CQ minimization are NP-complete [12], this is not a major problem for real-life applications, as the input (the CQ) is small.

It is known, on the other hand, that semantic information about the data, in the form of integrity constraints, alleviates query optimization by reducing the space of possible reformulations. In the previous analysis, however, constraints play no role, as CQ equivalence is defined over *all* databases. Adding constraints yields a refined notion of CQ equivalence, which holds over those databases that satisfy a given set of constraints only. But finding a minimal equivalent CQ in this context is notoriously more difficult than before. This is because basic static analysis tasks such as containment become undecidable when considered in full generality. This motivated a long research program for finding larger "islands of decidability" of such containment problem, based on syntactical restrictions on constraints [2, 8, 10, 11, 22, 23].

An important shortcoming of the previous approach, however, is that there is no theoretical guarantee that the minimized version of a CQ is in fact easier to evaluate (recall that, in general, CQ evaluation is NP-complete [12]). We know, on the other hand, quite a bit about classes of CQs that can be evaluated efficiently. It is thus a natural problem to ask whether constraints can be used to reformulate a CQ as one in such tractable classes, and if so, what is the cost of computing such reformulation. Following Abiteboul et al., this would provide us with a theoretical guarantee of "true efficiency" for those reformulations. We focus on one of the oldest and most studied tractability conditions for CQs; namely, *acyclicity*. It is known that acyclic CQs can be evaluated in linear time [27].

More formally, let us write $q \equiv_\Sigma q'$ whenever CQs $q$ and $q'$ are equivalent over all databases that satisfy $\Sigma$. In this work we study the following problem:

| PROBLEM : | SEMANTIC ACYCLICITY |
|---|---|
| INPUT : | A CQ $q$ and a finite set $\Sigma$ of constraints. |
| QUESTION : | Is there an acyclic CQ $q'$ s.t. $q \equiv_\Sigma q'$? |

We study this problem for the two most important classes of database constraints; namely:

1. *Tuple-generating dependencies* (tgds), i.e., expressions of the form $\forall \bar{x} \forall \bar{y}(\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\psi(\bar{x}, \bar{z}))$, where $\phi$ and $\psi$ are conjunctions of atoms. Tgds subsume the important class of referential integrity constraints (or inclusion dependencies).

2. *Equality-generating dependencies* (egds), i.e., expressions of the form $\forall \bar{x}(\phi(\bar{x}) \rightarrow y = z)$, where $\phi$ is a conjunction of atoms and $y, z$ are variables in $\bar{x}$. Egds subsume keys and functional dependencies (FDs).

A useful aspect of tgds and egds is that containment under them can be studied in terms of the *chase procedure* [25].

Coming back to semantic acyclicity, the main problem we study is, of course, decidability. Since basic reasoning with tgds and egds

is, in general, undecidable, we cannot expect semantic acyclicity to be decidable for arbitrary such constraints. Thus, we concentrate on the following question:

*Decidability:* For which classes of tgds and egds is the problem of semantic acyclicity decidable? In such cases, what is the computational cost of the problem?

Since semantic acyclicity is defined in terms of CQ equivalence under constraints, and the latter has received a lot of attention, it is relevant also to study the following question:

*Relationship to CQ equivalence:* What is the relationship between CQ equivalence and semantic acyclicity under constraints? Is the latter decidable for each class of tgds and egds for which the former is decidable?

Notice that if this was the case, one could transfer the mature theory of CQ equivalence under tgds and egds to tackle the problem of semantic acyclicity.

Finally, we want to understand to what extent semantic acyclicity helps CQ evaluation. Although an acyclic reformulation of a CQ can be evaluated efficiently, computing such reformulation might be expensive. Thus, it is relevant to study the following question:

*Evaluation:* What is the computational cost of evaluating semantically acyclic CQs under constraints?

**Semantic acyclicity in the absence of constraints.** The semantic acyclicity problem in the absence of dependencies (i.e., checking whether a CQ $q$ is equivalent to an acyclic one over the set of all databases) is by now well-understood. Regarding decidability, it is easy to prove that a CQ $q$ is semantically acyclic iff its core $q'$ is acyclic. (Recall that such $q'$ is the minimal equivalent CQ to $q$). It follows that checking semantic acyclicity in the absence of constraints is NP-complete (see, e.g., [6]). Regarding evaluation, semantically acyclic CQs can be evaluated efficiently [13, 14, 19].

**The relevance of constraints.** In the absence of constraints a CQ $q$ is equivalent to an acyclic one iff its core $q'$ is acyclic. Thus, the only reason why $q$ is not acyclic in the first hand is because it has not been minimized. This tells us that in this context semantic acyclicity is not really different from usual minimization. The presence of constraints, on the other hand, yields a more interesting notion of semantic acyclicity. This is because constraints can be applied on CQs to produce acyclic reformulations of them.

*Example 1.* This simple example helps understanding the role of tgds when reformulating CQs as acyclic ones. Consider a database that stores information about customers, records, and musical styles. The relation `Interest` contains pairs $(c, s)$ such that customer $c$ has declared interest in style $s$. The relation `Class` contains pairs $(r, s)$ such that record $r$ is of style $s$. Finally, the relation `Owns` contains a pair $(c, r)$ when customer $c$ owns record $r$.

Consider now a CQ $q(x, y)$ defined as follows:

$$\exists z \big( \texttt{Interest}(x, z) \wedge \texttt{Class}(y, z) \wedge \texttt{Owns}(x, y) \big).$$

This query asks for pairs $(c, r)$ such that customer $c$ owns record $r$ and has expressed interest in at least one of the styles with which $r$ is associated. This CQ is a core but it is not acyclic. Thus, from our previous observations it is not equivalent to an acyclic CQ (in the absence of constraints).

Assume now that we are told that this database contains compulsive music collectors only. In particular, each customer owns every record that is classified with a style in which he/she has expressed interest. This means that the database satisfies the tgd:

$$\tau = \texttt{Interest}(x, z), \texttt{Class}(y, z) \rightarrow \texttt{Owns}(x, y).$$

With this information at hand, we can easily reformulate $q(x, y)$ as the following acyclic CQ $q'(x, y)$:

$$\exists z \big( \texttt{Interest}(x, z) \wedge \texttt{Class}(y, z) \big).$$

Notice that $q$ and $q'$ are in fact equivalent over every database that satisfies $\tau$. ∎

**Contributions.** We observe that semantic acyclicity under constraints is not only more powerful, but also theoretically more challenging than in the absence of them. We start by studying decidability. In the process we also clarify the relationship between CQ equivalence and semantic acyclicity.

*Results for tgds:* Having a decidable CQ containment problem is a necessary condition for semantic acyclicity to be decidable under tgds.[1] Surprisingly enough, it is not a sufficient condition. This means that, contrary to what one might expect, there are natural classes of tgds for which CQ containment but not semantic acyclicity is decidable. In particular, this is the case for the well-known class of *full* tgds (i.e., tgds without existentially quantified variables in the head). In conclusion, we cannot directly export techniques from CQ containment to deal with semantic acyclicity.

In view of the previous results, we concentrate on classes of tgds that (a) have a decidable CQ containment problem, and (b) do not contain the class of full tgds. These restrictions are satisfied by several expressive languages considered in the literature. Such languages can be classified into three main families depending on the techniques used for studying their containment problem: (i) *guarded* tgds [8], which contain inclusion and linear dependencies, (ii) *non-recursive* [16], and (iii) *sticky* sets of tgds [10]. Instead of studying such languages one by one, we identify two semantic criteria that yield decidability for the semantic acyclicity problem, and then show that each one of the languages satisfies one such criteria.

- The first criterion is *acyclicity-preserving chase*. This is satisfied by those tgds for which the application of the chase over an acyclic instance preserves acyclicity. Guarded tgds enjoy this property. We establish that semantic acyclicity under guarded tgds is decidable and has the same complexity than its associated CQ containment problem: 2ExpTime-complete, and NP-complete for a fixed schema.

- The second criterion is *rewritability by unions of CQs (UCQs)*. Intuitively, a class $\mathbb{C}$ of sets of tgds has this property if the CQ containment problem under a set in $\mathbb{C}$ can always be reduced to a UCQ containment problem without constraints. Non-recursive and sticky sets of tgds enjoy this property. In the first case the complexity matches that of its associated CQ containment problem: NExpTime-complete, and NP-complete if the schema is fixed. In the second case, we get a NExpTime upper bound and an ExpTime lower bound. For a fixed schema the problem is NP-complete.

The NP bounds (under a fixed schema) can be seen as positive results: By spending exponential time in the size of the (small) query, we can not only minimize it using known techniques but also find an acyclic reformulation if one exists.

*Results for egds:* After showing that the techniques developed for tgds cannot be applied for showing the decidability of semantic acyclicity under egds, we focus on the class of keys over unary and binary predicates and we establish a positive result, namely semantic acyclicity is NP-complete. We prove this by showing

---

[1]Modulo some mild technical assumptions elaborated in the paper.

that in such context keys have acyclicity-preserving chase. Interestingly, this positive result can be extended to unary functional dependencies (over unconstrained signatures); this result has been established independently by Figueira [17]. We leave open whether the problem of semantic acyclicity under arbitrary egds, or even keys over arbitrary schemas, is decidable.

*Evaluation:* For tgds for which semantic acyclicity is decidable (guarded, non-recursive, sticky), we can use the following algorithm to evaluate a semantically acyclic CQ $q$ over a database $D$ that satisfies the constraints $\Sigma$:

1. Convert $q$ into an equivalent acyclic CQ $q'$ under $\Sigma$.
2. Evaluate $q'$ on $D$.
3. Return $q(D) = q'(D)$.

The running time is $O(|D| \cdot f(|q|, |\Sigma|))$, where $f$ is a double-exponential function (since $q'$ can be computed in double-exponential time for each one of the classes mentioned above and acyclic CQs can be evaluated in linear time). This constitutes a *fixed-parameter tractable algorithm* for evaluating $q$ on $D$. No such algorithm is believed to exist for CQ evaluation [26]; thus, semantically acyclic CQs under these constraints behave better than the general case in terms of evaluation.

But in the absence of constraints one can do better: Evaluating semantically acyclic CQs in such context is in polynomial time. It is natural to ask if this also holds in the presence of constraints. This is the case for guarded tgds and (arbitrary) FDs. For the other classes of constraints the problem remains to be investigated.

*Further results:* The results mentioned above continue to hold for a more "liberal" notion based on UCQs, i.e., checking whether a UCQ is equivalent to an acyclic union of CQs under the decidable classes of constraints identified above. Moreover, in case that a CQ $q$ is not equivalent to an acyclic CQ $q'$ under a set of constraints $\Sigma$, our proof techniques yield an *approximation of q under $\Sigma$* [4], that is, an acyclic CQ $q'$ that is maximally contained in $q$ under $\Sigma$. Computing and evaluating such approximation yields "quick" answers to $q$ when exact evaluation is infeasible.

**Finite vs. infinite databases.** The results mentioned above interpret the notion of CQ equivalence (and, thus, semantic acyclicity) over the set of both *finite* and *infinite* databases. The reason is the wide application of the chase we make in our proofs, which characterizes CQ equivalence under arbitrary databases only. This does not present a serious problem though, as all the particular classes of tgds for which we prove decidability in the paper (i.e., guarded, non-recursive, sticky) are *finitely controllable* [3, 18]. This means that CQ equivalence under arbitrary databases and under finite databases coincide. In conclusion, the results we obtain for such classes can be directly exported to the finite case.

**Organization.** Preliminaries are in Section 2. In Section 3 we consider semantic acyclicity under tgds. Acyclicity-preserving chase is studied in Section 4, and UCQ-rewritability in Section 5. Semantic acyclicity under egds is investigated in Section 6. Evaluation of semantically acyclic CQs is in Section 7. Finally, we present further advancements in Section 8 and conclusions in Section 9.

## 2. PRELIMINARIES

**Databases and conjunctive queries.** Let $\mathbf{C}$, $\mathbf{N}$ and $\mathbf{V}$ be disjoint countably infinite sets of *constants*, *(labeled) nulls* and (regular) *variables* (used in queries and dependencies), respectively, and $\sigma$ a relational schema. An *atom* over $\sigma$ is an expression of the form $R(\bar{v})$, where $R$ is a relation symbol in $\sigma$ of arity $n > 0$ and $\bar{v}$ is

an $n$-tuple over $\mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$. An *instance* over $\sigma$ is a (possibly infinite) set of atoms over $\sigma$ that contain constants and nulls, while a *database* over $\sigma$ is simply a finite instance over $\sigma$.

One of the central notions in our work is acyclicity. An instance $I$ is acyclic if it admits a *join tree*; i.e., if there exists a tree $T$ and a mapping $\lambda$ that associates with each node $t$ of $T$ an atom $\lambda(t)$ of $I$, such that the following holds:

1. For each atom $R(\bar{v})$ in $I$ there is a node $t$ in $T$ such that $\lambda(t) = R(\bar{v})$; and

2. For each null $x$ occurring in $I$ it is the case that the set $\{t \mid x \in \lambda(t)\}$ is connected in $T$.

A *conjunctive query* (CQ) over $\sigma$ is a formula of the form:

$$q(\bar{x}) := \exists \bar{y}\big(R_1(\bar{v}_1) \wedge \cdots \wedge R_m(\bar{v}_m)\big), \qquad (1)$$

where each $R_i(\bar{v}_i)$ $(1 \le i \le m)$ is an atom without nulls over $\sigma$, each variable mentioned in the $\bar{v}_i$'s appears either in $\bar{x}$ or $\bar{y}$, and $\bar{x}$ are the free variables of $q$. If $\bar{x}$ is empty, then $q$ is a *Boolean CQ*. As usual, the evaluation of CQs is defined in terms of *homomorphisms*. Let $I$ be an instance and $q(\bar{x})$ a CQ of the form (1). A homomorphism from $q$ to $I$ is a mapping $h$, which is the identity on $\mathbf{C}$, from the variables and constants in $q$ to the set of constants and nulls $\mathbf{C} \cup \mathbf{N}$ such that $R_i(h(\bar{v}_i)) \in I$,[2] for each $1 \le i \le m$. The *evaluation of $q(\bar{x})$ over $I$*, denoted $q(I)$, is the set of all tuples $h(\bar{x})$ over $\mathbf{C} \cup \mathbf{N}$ such that $h$ is a homomorphism from $q$ to $I$.

It is well-known that *CQ evaluation*, i.e., the problem of determining if a particular tuple $\bar{t}$ belongs to the evaluation $q(D)$ of a CQ $q$ over a database $D$, is NP-complete [12]. On the other hand, CQ evaluation becomes tractable by restricting the syntactic shape of CQs. One of the oldest and most common such restrictions is *acyclicity*. Formally, a CQ $q$ is acyclic if the instance consisting of the atoms of $q$ (after replacing each variable in $q$ with a fresh null) is acyclic. It is known from the seminal work of Yannakakis [27], that the problem of evaluating an acyclic CQ $q$ over a database $D$ can be solved in linear time $O(|q| \cdot |D|)$.

**Tgds and the chase procedure.** A *tuple-generating dependency* (tgd) over $\sigma$ is an expression of the form:

$$\forall \bar{x} \forall \bar{y}\big(\phi(\bar{x}, \bar{y}) \to \exists \bar{z} \psi(\bar{x}, \bar{z})\big), \qquad (2)$$

where both $\phi$ and $\psi$ are conjunctions of atoms without nulls over $\sigma$. For simplicity, we write this tgd as $\phi(\bar{x}, \bar{y}) \to \exists \bar{z} \psi(\bar{x}, \bar{z})$, and use comma instead of $\wedge$ for conjoining atoms. Further, we assume that each variable in $\bar{x}$ is mentioned in some atom of $\psi$. We call $\phi$ and $\psi$ the *body* and *head* of the tgd, respectively. The tgd in (2) is logically equivalent to the expression $\forall \bar{x}(q_\phi(\bar{x}) \to q_\psi(\bar{x}))$, where $q_\phi(\bar{x})$ and $q_\psi(\bar{x})$ are the CQs $\exists \bar{y}\phi(\bar{x}, \bar{y})$ and $\exists \bar{z}\psi(\bar{x}, \bar{z})$, respectively. Thus, an instance $I$ over $\sigma$ satisfies this tgd if and only if $q_\phi(I) \subseteq q_\psi(I)$. We say that an instance $I$ satisfies a set $\Sigma$ of tgds, denoted $I \models \Sigma$, if $I$ satisfies every tgd in $\Sigma$.

The *chase* is a useful tool when reasoning with tgds [8, 16, 22, 25]. We start by defining a single chase step. Let $I$ be an instance over schema $\sigma$ and $\tau = \phi(\bar{x}, \bar{y}) \to \exists \bar{z} \psi(\bar{x}, \bar{z})$ a tgd over $\sigma$. We say that $\tau$ is *applicable* w.r.t. $I$ if there exists a tuple $(\bar{a}, \bar{b})$ of elements in $I$ such that $\phi(\bar{a}, \bar{b})$ holds in $I$. In this case, *the result of applying $\tau$ over $I$ with $(\bar{a}, \bar{b})$* is the instance $J$ that extends $I$ with every atom in $\psi(\bar{a}, \bar{z}')$, where $\bar{z}'$ is the tuple obtained by simultaneously replacing each variable $z \in \bar{z}$ with a fresh distinct null not occurring in $I$. For such a single chase step we write $I \xrightarrow{\tau, (\bar{a}, \bar{b})} J$.

---

[2]As usual, we write $h(v_1, \dots, v_n)$ for $(h(v_1), \dots, h(v_n))$.

Let us assume now that $I$ is an instance and $\Sigma$ a finite set of tgds. A *chase sequence for $I$ under $\Sigma$* is a sequence:

$$I_0 \xrightarrow{\tau_0, \bar{c}_0} I_1 \xrightarrow{\tau_1, \bar{c}_1} I_2 \ldots$$

of chase steps such that: (1) $I_0 = I$; (2) For each $i \geq 0$, $\tau_i$ is a tgd in $\Sigma$; and (3) $\bigcup_{i \geq 0} I_i \models \Sigma$. We call $\bigcup_{i \geq 0} I_i$ the *result* of this chase sequence, which always exists. Although the result of a chase sequence is not necessarily unique (up to isomorphism), each such result is equally useful for our purposes since it can be homomorphically embedded into every other result. Thus, from now on, we denote by $chase(I, \Sigma)$ the result of an arbitrary chase sequence for $I$ under $\Sigma$. Further, for a CQ $q = \exists \bar{y}(R_1(\bar{v}_1) \wedge \cdots \wedge R_m(\bar{v}_m))$, we denote by $chase(q, \Sigma)$ the result of a chase sequence for the database $\{R_1(\bar{v}_1'), \ldots, R_m(\bar{v}_m')\}$ under $\Sigma$ obtained after replacing each variable $x$ in $q$ with a fresh constant $c(x)$.

**Egds and the chase procedure.** An *equality-generating dependency* (egd) over $\sigma$ is an expression of the form:

$$\forall \bar{x}(\phi(\bar{x}) \rightarrow x_i = x_j),$$

where $\phi$ is a conjunction of atoms without nulls over $\sigma$, and $x_i, x_j \in \bar{x}$. For clarity, we write this egd as $\phi(\bar{x}) \rightarrow x_i = x_j$, and use comma for conjoining atoms. We call $\phi$ the *body* of the egd. An instance $I$ over $\sigma$ satisfies this egd if, for every homomorphism $h$ such that $h(\phi(\bar{x})) \subseteq I$, it is the case that $h(x_i) = h(x_j)$. An instance $I$ satisfies a set $\Sigma$ of egds, denoted $I \models \Sigma$, if $I$ satisfies every egd in $\Sigma$.

Recall that egds subsume functional dependencies, which in turn subsume keys. A *functional dependency* (FD) over $\sigma$ is an expression of the form $R : A \rightarrow B$, where $R$ is a relation symbol in $\sigma$ of arity $n > 0$, and $A, B$ are subsets of $\{1, \ldots, n\}$, asserting that the values of the attributes of $B$ are determined by the values of the attributes of $A$. For example, $R : \{1\} \rightarrow \{3\}$, where $R$ is a ternary relation, is actually the egd $R(x, y, z), R(x, y', z') \rightarrow z = z'$. A FD $R : A \rightarrow B$ as above is called *key* if $A \cup B = \{1, \ldots, n\}$.

As for tgds, the chase is a useful tool when reasoning with egds. Let us first define a single chase step. Consider an instance $I$ over schema $\sigma$ and an egd $\epsilon = \phi(\bar{x}) \rightarrow x_i = x_j$ over $\sigma$. We say that $\epsilon$ is *applicable* w.r.t. $I$ if there exists a homomorphism $h$ such that $h(\phi(\bar{x})) \subseteq I$ and $h(x_i) \neq h(x_j)$. In this case, *the result of applying $\epsilon$ over $I$ with $h$* is as follows: If both $h(x_i), h(x_j)$ are constants, then the result is "failure"; otherwise, it is the instance $J$ obtained from $I$ by identifying $h(x_i)$ and $h(x_j)$ as follows: If one is a constant, then every occurrence of the null is replaced by the constant, and if both are nulls, the one is replaced everywhere by the other. As for tgds, we can define the notion of the chase sequence for an instance $I$ under a set $\Sigma$ of egds. Notice that such a sequence, assuming that is not failing, always is finite. Moreover, it is unique (up to null renaming), and thus we refer to *the* chase for $I$ under $\Sigma$, denoted $chase(I, \Sigma)$. Further, for a CQ $q = \exists \bar{y}(R_1(\bar{v}_1) \wedge \cdots \wedge R_m(\bar{v}_m))$, we denote by $chase(q, \Sigma)$ the result of a chase sequence for the database $\{R_1(\bar{v}_1'), \ldots, R_m(\bar{v}_m')\}$ under $\Sigma$ obtained after replacing each variable $x$ in $q$ with a fresh constant $c(x)$; however, it is important to clarify that these are special constants, which are treated as nulls during the chase.

**Containment and equivalence.** Let $q$ and $q'$ be CQs and $\Sigma$ a finite set of tgds or egds. Then, $q$ is *contained* in $q'$ under $\Sigma$, denoted $q \subseteq_\Sigma q'$, if $q(I) \subseteq q'(I)$ for every instance $I$ such that $I \models \Sigma$. Further, $q$ is *equivalent* to $q'$ under $\Sigma$, denoted $q \equiv_\Sigma q'$, whenever $q \subseteq_\Sigma q'$ and $q' \subseteq_\Sigma q$ (or, equivalently, if $q(I) = q'(I)$ for every instance $I$ such that $I \models \Sigma$). The following well-known charac-

terization of CQ containment in terms of the chase will be widely used in our proofs:

LEMMA 1. *Let $q(\bar{x})$ and $q'(\bar{x}')$ be CQs and $\Sigma$ be a finite set of tgds or egds. Then $q \subseteq_\Sigma q'$ if and only if $c(\bar{x})$ belongs to the evaluation of $q'$ over $chase(q, \Sigma)$.*

A problem that is quite important for our work is *CQ containment under constraints* (tgds or egds), defined as follows: Given CQs $q, q'$ and a finite set $\Sigma$ of tgds or egds, is it the case that $q \subseteq_\Sigma q'$? Whenever $\Sigma$ is bound to belong to a particular class $\mathbb{C}$ of sets of tgds, we denote this problem as $\mathsf{Cont}(\mathbb{C})$. It is clear that the above lemma provides a decision procedure for the containment problem under egds. However, this is not the case for tgds.

**Decidable containment of CQs under tgds.** It is not surprising that Lemma 1 does not provide a decision procedure for solving CQ containment under tgds since this problem is known to be undecidable [7]. This has led to a flurry of activity for identifying syntactic restrictions on sets of tgds that lead to decidable CQ containment (even in the case when the chase does not terminate).[3] Such restrictions are often classified into three main paradigms:

*Guardedness:* A tgd is *guarded* if its body contains an atom, called *guard*, that contains all the body-variables. Although the chase under guarded tgds does not necessarily terminate, query containment is decidable. This follows from the fact that the result of the chase has *bounded treewidth*. Let $\mathbb{G}$ be the class of sets of guarded tgds.

PROPOSITION 2. [8] $\mathsf{Cont}(\mathbb{G})$ *is 2EXPTIME-complete. It becomes EXPTIME-complete if the arity of the schema is fixed, and NP-complete if the schema is fixed.*

A key subclass of guarded tgds is the class of *linear* tgds, that is, tgds whose body consists of a single atom [9], which in turn subsume the well-known class of *inclusion dependencies* (linear tgds without repeated variables neither in the body nor in the head) [15]. Let $\mathbb{L}$ and $\mathbb{ID}$ be the classes of sets of linear tgds and inclusions dependencies, respectively. $\mathsf{Cont}(\mathbb{C})$, for $\mathbb{C} \in \{\mathbb{L}, \mathbb{ID}\}$, is PSPACE-complete, and NP-complete if the arity of the schema is fixed [22].

*Non-recursiveness:* A set $\Sigma$ of tgds is *non-recursive* if its predicate graph contains no directed cycles. (Non-recursive sets of tgds are also known as *acyclic* [16, 24], but we reserve this term for CQs). Non-recursiveness ensures the termination of the chase, and thus decidability of CQ containment. Let $\mathbb{NR}$ be the class of non-recursive sets of tgds. Then:

PROPOSITION 3. [24] $\mathsf{Cont}(\mathbb{NR})$ *is complete for NEXPTIME, even if the arity of the schema is fixed. It becomes NP-complete if the schema is fixed.*

*Stickiness:* This condition ensures neither termination nor bounded treewidth of the chase. Instead, the decidability of query containment is obtained by exploiting *query rewriting* techniques. The goal of stickiness is to capture joins among variables that are not expressible via guarded tgds, but without forcing the chase to terminate. The key property underlying this condition can be described as follows: During the chase, terms that are associated (via a homomorphism) with variables that appear more than once in the body of a tgd (i.e., join variables) are always propagated (or "stick") to the inferred atoms. This is illustrated in Figure 1(a); the first set of tgds

---

[3]In fact, these restrictions are designed to obtain decidable *query answering under tgds*. However, this problem is equivalent to query containment under tgds (Lemma 1).
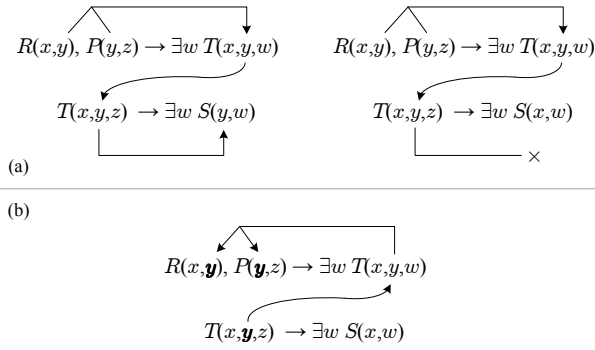
$$R(x,y), P(y,z) \rightarrow \exists w \, T(x,y,w) \qquad R(x,y), P(y,z) \rightarrow \exists w \, T(x,y,w)$$

$$T(x,y,z) \rightarrow \exists w \, S(y,w) \qquad\qquad T(x,y,z) \rightarrow \exists w \, S(x,w)$$

(a)

(b)

$$R(x,\boldsymbol{y}), P(\boldsymbol{y},z) \rightarrow \exists w \, T(x,y,w)$$

$$T(x,\boldsymbol{y},z) \rightarrow \exists w \, S(x,w)$$

**Figure 1: Stickiness and marking.**

is sticky, while the second is not. The formal definition is based on an inductive marking procedure that marks the variables that may violate the semantic property of the chase described above [10]. Roughly, during the base step of this procedure, a variable that appears in the body of a tgd $\tau$ but not in every head-atom of $\tau$ is marked. Then, the marking is inductively propagated from head to body as shown in Figure 1(b). Finally, a finite set of tgds $\Sigma$ is *sticky* if no tgd in $\Sigma$ contains two occurrences of a marked variable. Then:

PROPOSITION 4. [10] $\mathsf{Cont}(\mathbb{S})$ *is* EXPTIME-*complete. It becomes* NP-*complete if the arity of the schema is fixed.*

*Weak versions:* Each one of the previous classes has an associated weak version, called *weakly-guarded* [8], *weakly-acyclic* [16], and *weakly-sticky* [10], respectively, that guarantees the decidability of query containment. The underlying idea of all these classes is the same: Relax the conditions in the definition of the class, so that only those positions that receive null values during the chase procedure are taken into consideration. A key property of all these classes is that they extend the class of *full tgds*, i.e., those without existentially quantified variables. This is not the case for the "unrelaxed" versions presented above.

## 3. SEMANTIC ACYCLICITY WITH TGDS

One of the main tasks of our work is to study the problem of checking whether a CQ $q$ is equivalent to an acyclic CQ over those instances that satisfy a set $\Sigma$ of tgds. When this is the case we say that $q$ is *semantically acyclic under* $\Sigma$. The semantic acyclicity problem is defined below; $\mathbb{C}$ is a class of sets of tgds (e.g., guarded, non-recursive, sticky, etc.):

| | |
|---|---|
| PROBLEM : | $\mathsf{SemAc}(\mathbb{C})$ |
| INPUT : | A CQ $q$ and a finite set $\Sigma$ of tgds in $\mathbb{C}$. |
| QUESTION : | Is there an acyclic CQ $q'$ s.t. $q \equiv_\Sigma q'$? |

### 3.1 Infinite Instances vs. Finite Databases

It is important to clarify that $\mathsf{SemAc}(\mathbb{C})$ asks for the existence of an acyclic CQ $q'$ that is equivalent to $q$ under $\Sigma$ focussing on arbitrary (finite or infinite) instances. However, in practice we are concerned only with finite databases. Therefore, one may claim that the natural problem to investigate is $\mathsf{FinSemAc}(\mathbb{C})$, which accepts as input a CQ $q$ and a finite set $\Sigma \in \mathbb{C}$ of tgds, and asks whether an acyclic CQ $q'$ exists such that $q(D) = q'(D)$ for every finite database $D \models \Sigma$.

Interestingly, for all the classes of sets of tgds discussed in the previous section, $\mathsf{SemAc}$ and $\mathsf{FinSemAc}$ coincide due to the fact that they ensure the so-called *finite controllability* of CQ containment. This means that query containment under arbitrary instances and query containment under finite databases are equivalent problems. For non-recursive and weakly-acyclic sets of tgds this immediately follows from the fact that the chase terminates. For guarded-based classes of sets of tgds this has been shown in [3], while for sticky-based classes of sets of tgds it has been shown in [18]. Therefore, assuming that $\mathbb{C}$ is one of the above syntactic classes of sets of tgds, by giving a solution to $\mathsf{SemAc}(\mathbb{C})$ we immediately obtain a solution for $\mathsf{FinSemAc}(\mathbb{C})$.

The reason why we prefer to focus on $\mathsf{SemAc}(\mathbb{C})$, instead of $\mathsf{FinSemAc}(\mathbb{C})$, is given by Lemma 1: Query containment under arbitrary instances can be characterized in terms of the chase. This is not true for CQ containment under finite databases simply because the chase is, in general, infinite.

### 3.2 Semantic Acyclicity vs. Containment

There is a close relationship between semantic acyclicity and a restricted version of CQ containment under sets of tgds, as we explain next. But first we need to recall the notion of connectedness for queries and tgds. A CQ is *connected* if its *Gaifman graph* is connected – recall that the nodes of the Gaifman graph of a CQ $q$ are the variables of $q$, and there is an edge between variables $x$ and $y$ iff they appear together in some atom of $q$. Analogously, a tgd $\tau$ is *body-connected* if its body is connected. Then:

PROPOSITION 5. *Let* $\Sigma$ *be a finite set of body-connected tgds and* $q, q'$ *two Boolean and connected CQs without common variables, such that* $q$ *is acyclic and* $q'$ *is not semantically acyclic under* $\Sigma$. *Then* $q \subseteq_\Sigma q'$ *iff* $q \wedge q'$ *is semantically acyclic under* $\Sigma$.

As an immediate corollary of Proposition 5, we obtain an initial boundary for the decidability of $\mathsf{SemAc}$: We can only obtain a positive result for those classes of sets of tgds for which the restricted containment problem presented above is decidable. More formally, let us define $\mathsf{RestCont}(\mathbb{C})$ to be the problem of checking $q \subseteq_\Sigma q'$, given a set $\Sigma$ of body-connected tgds in $\mathbb{C}$ and two Boolean and connected CQs $q$ and $q'$, without common variables, such that $q$ is acyclic and $q'$ is not semantically acyclic under $\Sigma$. Then:

COROLLARY 6. $\mathsf{SemAc}(\mathbb{C})$ *is undecidable for every class* $\mathbb{C}$ *of tgds such that* $\mathsf{RestCont}(\mathbb{C})$ *is undecidable.*

As we shall discuss later, $\mathsf{RestCont}$ is not easier than general CQ containment under tgds, which means that the only classes of tgds for which we know the former problem to be decidable are those for which we know CQ containment to be decidable (e.g., those introduced in Section 2).

At this point, one might be tempted to think that some version of the converse of Proposition 5 also holds; that is, the semantic acyclicity problem for $\mathbb{C}$ is reducible to the containment problem for $\mathbb{C}$. This would imply the decidability of $\mathsf{SemAc}$ for any class of sets of tgds for which the CQ containment problem is decidable. Our next result shows that the picture is more complicated than this as $\mathsf{SemAc}$ is undecidable even over the class $\mathbb{F}$ of sets of full tgds, which ensures the decidability of CQ containment:

THEOREM 7. *The problem* $\mathsf{SemAc}(\mathbb{F})$ *is undecidable.*

PROOF. We provide a sketch since the complete construction is long. We reduce from the *Post correspondence problem* (PCP) over the alphabet $\{a, b\}$. The input to this problem are two equally long lists $w_1, \ldots, w_n$ and $w'_1, \ldots, w'_n$ of words over $\{a, b\}$, and we ask whether there is a *solution*, i.e., a nonempty sequence $i_1 \ldots i_m$ of indices in $\{1, \ldots, n\}$ such that $w_{i_1} \ldots w_{i_m} = w'_{i_1} \ldots w'_{i_m}$.
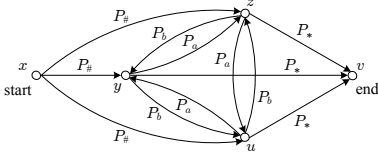
**Figure 2: The query $q$ from the proof of Theorem 7.**

Let $w_1, \ldots, w_n$ and $w_1', \ldots, w_n'$ be an instance of PCP. In the full proof we construct a Boolean CQ $q$ and a set $\Sigma$ of full tgds over the signature $\{P_a, P_b, P_\#, P_*, \text{sync}, \text{start}, \text{end}\}$, where $P_a$, $P_b$, $P_\#$, $P_*$ and sync are binary predicates, and start and end are unary predicates, such that the PCP instance given by $w_1, \ldots, w_n$ and $w_1', \ldots, w_n'$ has a solution iff there exists an acyclic CQ $q'$ such that $q \equiv_\Sigma q'$. In this sketch though, we concentrate on the case when the underlying graph of $q'$ is a directed path; i.e, we prove that the PCP instance has a solution iff there is a CQ $q'$ whose underlying graph is a directed path such that $q \equiv_\Sigma q'$. This does not imply the undecidability of the general case, but the proof of the latter is a generalization of the one we sketch below.

The restriction of the query $q$ to the symbols that are not sync is graphically depicted in Figure 2. There, $x, y, z, u, v$ denote the names of the respective variables. The interpretation of sync in $q$ consists of all pairs in $\{y, u, z\}$.

Our set $\Sigma$ of full tgds defines the *synchronization* predicate sync over those acyclic CQs $q'$ whose underlying graph is a path. Assume that $q'$ encodes a word $w \in \{a, b\}^+$. We denote by $w[i]$, for $1 \le i \le |w|$, the prefix of $w$ of length $i$. In such case, the predicate sync contains those pairs $(i, j)$ such that for some sequence $i_1 \ldots i_m$ of indices in $\{1, \ldots, n\}$ we have that $w_{i_1} \ldots w_{i_m} = w[i]$ and $w_{i_1}' \ldots w_{i_m}' = w[j]$. Thus, if $w$ is a solution for the PCP instance, then $(|w|, |w|)$ belongs to the interpretation of sync.

Formally, $\Sigma$ consists of the following rules:

1. An initialization rule:

$$\text{start}(x), P_\#(x, y) \rightarrow \text{sync}(y, y).$$

   That is, the first element after the special symbol $\#$ (which denotes the beginning of a word over $\{a, b\}$) is synchronized with itself.

2. For each $1 \le i \le n$, a synchronization rule:

$$\text{sync}(x, y), P_{w_i}(x, z), P_{w_i'}(y, u) \rightarrow \text{sync}(z, u).$$

   Here, $P_w(x, y)$, for $w = a_1 \ldots a_t \in \{a, b\}^+$, denotes $P_{a_1}(x, x_1), \ldots, P_{a_t}(x_{t-1}, y)$, where the $x_i$'s are fresh variables. Roughly, if $(x, y)$ is synchronized and the element $z$ (resp., $u$) is reachable from $x$ (resp., $y$) by word $w_i$ (resp., $w_i'$), then $(z, u)$ is also synchronized.

3. For each $1 \le i \le n$, a finalization rule:

$$\text{start}(x), P_a(y, z), P_a(z, u), P_*(u, v), \text{end}(v),$$
$$\text{sync}(y_1, y_2), P_{w_i}(y_1, y), P_{w_i'}(y_2, y) \rightarrow \psi,$$

   where $\psi$ is the conjunction of atoms:

$$P_\#(x, y), P_\#(x, z), P_\#(x, u), P_*(y, v), P_*(z, v),$$
$$P_b(z, y), P_b(u, z), P_a(u, y), P_b(y, u),$$
$$\text{sync}(y, y), \text{sync}(z, z), \text{sync}(y, z), \text{sync}(z, y),$$
$$\text{sync}(y, u), \text{sync}(u, y), \text{sync}(z, u), \text{sync}(u, z).$$

This tgd enforces $chase(q', \Sigma)$ to contain a "copy" of $q$ whenever $q'$ encodes a solution for the PCP instance.

We first show that if the PCP instance has a solution given by the nonempty sequence $i_1 \ldots i_m$, with $1 \le i_1, \ldots, i_m \le n$, then there exists an acyclic CQ $q'$ whose underlying graph is a directed path such that $q \equiv_\Sigma q'$. Let us assume that $w_{i_1} \ldots w_{i_m} = a_1 \ldots a_t$, where each $a_i \in \{a, b\}$. It is not hard to prove that $q \equiv_\Sigma q'$, where $q'$ is as follows:
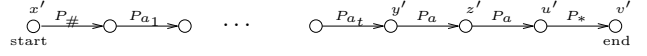


Here, again, $x', y', z', u', v'$ denote the names of the respective variables of $q'$. All nodes in the above path are different. The main reason why $q \equiv_\Sigma q'$ holds is because the fact $w$ is a solution implies that there are elements $y_1$ and $y_2$ such that $\text{sync}(y_1, y_2)$, $P_{w_1}(y_1, y)$ and $P_{w_i'}(y_2, y)$ hold in $chase(q', \Sigma)$. Thus, the finalization rule is fired. This creates a copy of $q$ in $chase(q', \Sigma)$, which allows $q$ to be homomorphically mapped to $chase(q', \Sigma)$.

Now we prove that if there exists an acyclic CQ $q'$ such that $q \equiv_\Sigma q'$ and the underlying graph of $q'$ is a directed path, then the PCP instance has a solution. Since $q \equiv_\Sigma q'$, Lemma 1 tells us that $chase(q, \Sigma) \equiv chase(q', \Sigma)$ are homomorphically equivalent. But then $chase(q', \Sigma)$ must contain at least one variable labeled start and one variable labeled end. The first variable cannot have incoming edges (otherwise, $chase(q', \Sigma)$ would not homomorphically map to $chase(q, \Sigma)$), while the second one cannot have outcoming edges (for the same reason). Thus, it is the first variable $x'$ of $q'$ that is labeled start and the last one $v'$ that is labeled end. Further, all edges reaching $v'$ in $q'$ must be labeled $P_*$ (otherwise $q'$ does not homomorphically map to $q$). Thus, this is the label of the last edge of $q'$ that goes from variable $u'$ to $v'$. Analogously, the edge that leaves $x'$ in $q'$ is labeled $P_\#$. Further, any other edge in $q'$ is labeled $P_a$, $P_b$, or sync.

Notice now that $v'$ must have an incoming edge labeled $P_*$ in $chase(q', \Sigma)$ from some node $u''$ that has an outgoing edge with label $P_a$ (since $q$ homomorphically maps to $chase(q', \Sigma)$). By definition of $\Sigma$, this could only have happened if the finalization rule is fired. In particular, $u'$ is preceded by node $z'$, which in turn is preceded by $y'$, and there are elements $y_1'$ and $y_2'$ such that $\text{sync}(y_1', y_2')$, $P_{w_1}(y_1', y')$ and $P_{w_i'}(y_2', y')$ hold in $chase(q', \Sigma)$. In fact, the unique path from $y_1'$ (resp., $y_2'$) to $y'$ in $q'$ is labeled $w_i$ (resp., $w_i'$). This means that the atom $\text{sync}(y_1', y_2')$ was not one of the edges of $q'$, but must have been produced during the chase by firing the initialization or the synchronization rules, and so on. This process must finish in the second element $x^*$ of $q'$. (Recall that $\text{sync}(x^*, x^*)$ belongs to $chase(q', \Sigma)$ due to the first rule of $\Sigma$.) We conclude that our PCP instance has a solution. $\square$

Theorem 7 rules out any class that captures the class of full tgds, e.g., weakly-guarded, weakly-acyclic and weakly-sticky sets of tgds. The question that comes up is whether the non-weak versions of the above classes, namely guarded, non-recursive and sticky sets of tgds, ensure the decidability of SemAc, and what is the complexity of the problem. This is the subject of the next two sections.

# 4. ACYCLICITY-PRESERVING CHASE

We propose a semantic criterion, the so-called *acyclicity-preserving chase*, that ensures the decidability of $\mathsf{SemAc}(\mathbb{C})$ whenever the problem $\mathsf{Cont}(\mathbb{C})$ is decidable. This criterion guarantees that, starting from an acyclic instance, it is not possible to destroy
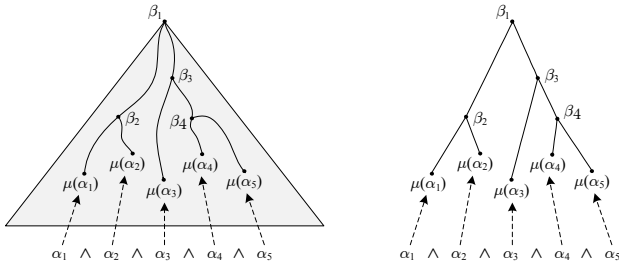
**Figure 3: The compact acyclic query.**

its acyclicity during the construction of the chase. We then proceed to show that the class of guarded sets of tgds has acyclicity-preserving chase, which immediately implies the decidability of SemAc($\mathbb{G}$), and we pinpoint the exact complexity of the latter problem. Notice that non-recursiveness and stickiness do not enjoy this property, even in the restrictive setting where only unary and binary predicates can be used; more details are given in the next section. The formal definition of our semantic criterion follows:

*Definition 1.* (**Acyclicity-preserving Chase**) We say that a class $\mathbb{C}$ of sets of tgds has *acyclicity-preserving chase* if, for every acyclic CQ $q$, set $\Sigma \in \mathbb{C}$, and chase sequence for $q$ under $\Sigma$, the result of such a chase sequence is acyclic. ∎

We can then prove the following small query property:

PROPOSITION 8. *Let $\Sigma$ be a finite set of tgds that belongs to a class that has acyclicity-preserving chase, and $q$ a CQ. If $q$ is semantically acyclic under $\Sigma$, then there exists an acyclic CQ $q'$, where $|q'| \leq 2 \cdot |q|$, such that $q \equiv_\Sigma q'$.*

The proof of the above result relies on the following technical lemma, established in [8] (using slightly different terminology), that will also be used later in our investigation:

LEMMA 9. *Let $q(\bar{x})$ be a CQ, $I$ an acyclic instance, and $\bar{c}$ a tuple of distinct constants occurring in $I$ such that $q(\bar{c})$ holds in $I$. There exists an acyclic CQ $q'(\bar{x})$, where $q' \subseteq q$ and $|q'| \leq 2 \cdot |q|$, such that $q'(\bar{c})$ holds in $I$.*

For the sake of completeness, we would like to recall the idea of the construction underlying Lemma 9, which is illustrated in Figure 3. Assuming that $\alpha_1, \ldots, \alpha_5$ are the atoms of $q$, there exists a homomorphism $\mu$ that maps $\alpha_1 \wedge \ldots \wedge \alpha_5$ to the join tree $T$ of the acyclic instance $I$ (the shaded tree in Figure 3). Consider now the subtree $T_q$ of $T$ consisting of all the nodes in the image of the query and their ancestors. From $T_q$ we extract the smaller tree $F$ also depicted in Figure 3; $F = (V, E)$ is obtained as follows:

1. $V$ consists of all the root and leaf nodes of $T_q$, and all the inner nodes of $T_q$ with at least two children; and
2. For every $v, u \in V$, $(v, u) \in E$ iff $u$ is a descendant of $v$ in $T_q$, and the only nodes of $V$ that occur on the unique shortest path from $v$ to $u$ in $T_q$ are $v$ and $u$.

It is easy to verify that $F$ is a join tree, and has at most $2 \cdot |q|$ nodes. The acyclic conjunctive query $q'$ is defined as the conjunction of all atoms occurring in $F$.

Notice that a result similar to Lemma 9 is implicit in [4], where the problem of approximating conjunctive queries is investigated. However, from the results of [4], we can only conclude the existence of an exponentially sized acyclic CQ in the arity of the underlying schema, while Lemma 9 establishes the existence of an

acyclic query of linear size. This is decisive for our later complexity analysis. Having the above lemma in place, it is not difficult to establish Proposition 8.

PROOF OF PROPOSITION 8. Since, by hypothesis, $q$ is semantically acyclic under $\Sigma$, there exists an acyclic CQ $q''(\bar{x})$ such that $q \equiv_\Sigma q''$. By Lemma 1, $c(\bar{x})$ belongs to the evaluation of $q$ over $chase(q'', \Sigma)$. Recall that $\Sigma$ belongs to a class that has acyclicity-preserving chase, which implies that $chase(q'', \Sigma)$ is acyclic. Hence, by Lemma 9, there exists an acyclic CQ $q'$, where $q' \subseteq q$ and $|q'| \leq 2 \cdot |q|$, such that $c(\bar{x})$ belongs to the evaluation of $q'$ over $chase(q'', \Sigma)$. By Lemma 1, $q'' \subseteq_\Sigma q'$, and therefore $q \subseteq_\Sigma q'$. We conclude that $q \equiv_\Sigma q'$, and the claim follows. □

It is clear that Proposition 8 provides a decision procedure for SemAc($\mathbb{C}$) whenever $\mathbb{C}$ has acyclicity-preserving chase and Cont($\mathbb{C}$) is decidable. Given a CQ $q$, and a finite set $\Sigma \in \mathbb{C}$:

1. Guess an acyclic CQ $q'$ of size at most $2 \cdot |q|$; and
2. Verify that $q \subseteq_\Sigma q'$ and $q' \subseteq_\Sigma q$.

The next result follows:

THEOREM 10. *Consider a class $\mathbb{C}$ of sets of tgds that has acyclicity-preserving chase. If the problem Cont($\mathbb{C}$) is decidable, then SemAc($\mathbb{C}$) is also decidable.*

## 4.1 Guardedness

We proceed to show that SemAc($\mathbb{G}$) is decidable and has the same complexity as CQ containment under guarded tgds:

THEOREM 11. SemAc($\mathbb{G}$) *is complete for* 2EXPTIME. *It becomes* EXPTIME-*complete if the arity of the schema is fixed, and* NP-*complete if the schema is fixed.*

The rest of this section is devoted to establish Theorem 11.

**Decidability and Upper Bounds**

We first show that:

PROPOSITION 12. $\mathbb{G}$ *has acyclicity-preserving chase.*

The above result, combined with Theorem 10, implies the decidability of SemAc($\mathbb{G}$). However, this does not say anything about the complexity of the problem. With the aim of pinpointing the exact complexity of SemAc($\mathbb{G}$), we proceed to analyze the complexity of the decision procedure underlying Theorem 10. Recall that, given a CQ $q$, and a finite set $\Sigma \in \mathbb{G}$, we guess an acyclic CQ $q'$ such that $|q'| \leq 2 \cdot |q|$, and verify that $q \equiv_\Sigma q'$. It is clear that this algorithm runs in non-deterministic polynomial time with a call to a $\mathcal{C}$ oracle, where $\mathcal{C}$ is a complexity class powerful enough for solving Cont($\mathbb{G}$). Thus, Proposition 2 implies that SemAc($\mathbb{G}$) is in 2EXPTIME, in EXPTIME if the arity of the schema is fixed, and in NP if the schema is fixed. One may ask why for a fixed schema the obtained upper bound is NP and not $\Sigma_2^P$. Observe that the oracle is called only once in order to solve Cont($\mathbb{G}$), and since Cont($\mathbb{G}$) is already in NP when the schema is fixed, it is not really needed in this case.

**Lower Bounds**

Let us now show that the above upper bounds are optimal. By Proposition 5, RestCont($\mathbb{G}$) can be reduced in constant time to SemAc($\mathbb{G}$). Thus, to obtain the desired lower bounds, it suffices to

reduce in polynomial time $\mathsf{Cont}(\mathbb{G})$ to $\mathsf{RestCont}(\mathbb{G})$. Interestingly, the lower bounds given in Section 2 for $\mathsf{Cont}(\mathbb{G})$ hold even if we focus on Boolean CQs and the left-hand side query is acyclic. In fact, this is true, not only for guarded, but also for non-recursive and sticky sets of tgds. Let $\mathsf{AcBoolCont}(\mathbb{C})$ be the following problem: Given an acyclic Boolean CQ $q$, a Boolean CQ $q'$, and a finite set $\Sigma \in \mathbb{C}$ of tgds, is it the case $q \subseteq_\Sigma q'$?

From the above discussion, to establish the desired lower bounds for guarded sets of tgds (and also for the other classes of tgds considered in this work), it suffices to reduce in polynomial time $\mathsf{AcBoolCont}$ to $\mathsf{RestCont}$. To this end, we introduce the so-called connecting operator, which provides a generic reduction from $\mathsf{AcBoolCont}$ to $\mathsf{RestCont}$.

**Connecting operator.** Consider an acyclic Boolean CQ $q$, a Boolean CQ $q'$, and a finite set $\Sigma$ of tgds. We assume that both $q, q'$ are of the form $\exists \bar{y}(R_1(\bar{v}_1) \wedge \cdots \wedge R_m(\bar{v}_m))$. The application of the *connecting operator* on $(q, q', \Sigma)$ returns the triple $(\mathsf{c}(q), \mathsf{c}(q'), \mathsf{c}(\Sigma))$, where

- $\mathsf{c}(q)$ is the query

$$\exists \bar{y}\exists w\big(R_1^\star(\bar{v}_1, w) \wedge \cdots \wedge R_m^\star(\bar{v}_m, w) \wedge aux(w, w)\big),$$

  where $w$ is a new variable not in $q$, each $R_i^\star$ is a new predicate, and also $aux$ is a new binary predicate;

- $\mathsf{c}(q')$ is the query

$$\exists \bar{y}\exists w\exists u\exists v\big(R_1^\star(\bar{v}_1, w) \wedge \cdots \wedge R_m^\star(\bar{v}_m, w) \wedge$$
$$aux(w, u) \wedge aux(u, v) \wedge aux(v, w)\big),$$

  where $w, u, v$ are new variables not in $q$; and

- Finally, $\mathsf{c}(\Sigma) = \{\mathsf{c}(\tau) \mid \tau \in \Sigma\}$, where for a tgd $\tau = \phi(\bar{x}, \bar{y}) \to \exists \bar{z}\psi(\bar{x}, \bar{z})$, $\mathsf{c}(\tau)$ is the tgd

$$\phi^\star(\bar{x}, \bar{y}, w) \to \exists \bar{z}\psi^\star(\bar{x}, \bar{z}, w),$$

  with $\phi^\star(\bar{x}, \bar{y}, w), \psi^\star(\bar{x}, \bar{z}, w)$ be the conjunctions obtained from $\phi(\bar{x}, \bar{y}), \psi(\bar{x}, \bar{z})$, respectively, by replacing each atom $R(x_1, \ldots, x_n)$ with $R^\star(x_1, \ldots, x_n, w)$, where $w$ is a new variable not occurring in $\tau$.

This concludes the definition of the connecting operator. A class $\mathbb{C}$ of sets of tgds is *closed under connecting* if, for every set $\Sigma \in \mathbb{C}$, $\mathsf{c}(\Sigma) \in \mathbb{C}$. It is easy to verify that $\mathsf{c}(q)$ remains acyclic and is connected, $\mathsf{c}(q')$ is connected and not semantically acyclic under $\mathsf{c}(\Sigma)$, and $\mathsf{c}(\Sigma)$ is a set of body-connected tgds. It can be also shown that $q \subseteq_\Sigma q'$ iff $\mathsf{c}(q) \subseteq_{\mathsf{c}(\Sigma)} \mathsf{c}(q')$.

From the above discussion, it is clear that the connecting operator provides a generic polynomial time reduction from $\mathsf{AcBoolCont}(\mathbb{C})$ to $\mathsf{RestCont}(\mathbb{C})$, for every class $\mathbb{C}$ of sets of tgds that is closed under connecting. Then:

PROPOSITION 13. *Let $\mathbb{C}$ be a class of sets of tgds that is closed under connecting such that $\mathsf{AcBoolCont}(\mathbb{C})$ is hard for a complexity class $\mathcal{C}$ that is closed under polynomial time reductions. Then, $\mathsf{SemAc}(\mathbb{C})$ is also $\mathcal{C}$-hard.*

**Back to guardedness.** It is easy to verify that the class of guarded sets of tgds is closed under connecting. Thus, the lower bounds for $\mathsf{SemAc}(\mathbb{G})$ stated in Theorem 11 follow from Propositions 2 and 13. Note that, although Proposition 2 refers to $\mathsf{Cont}(\mathbb{G})$, the lower bounds hold for $\mathsf{AcBoolCont}(\mathbb{G})$; this is implicit in [8].

As said in Section 2, a key subclass of guarded sets of tgds is the class of linear tgds, i.e., tgds whose body consists of a single atom,

which in turn subsume the well-known class of inclusion dependencies. By exploiting the non-deterministic procedure employed for $\mathsf{SemAc}(\mathbb{G})$, and the fact that both linear tgds and inclusion dependencies are closed under connecting, we can show that:

THEOREM 14. $\mathsf{SemAc}(\mathbb{C})$, *for $\mathbb{C} \in \{\mathbb{L}, \mathbb{ID}\}$, is complete for* PSPACE. *It becomes* NP-*complete if the arity of the schema is fixed.*

# 5. UCQ REWRITABILITY

Even though the acyclicity-preserving chase criterion was very useful for solving $\mathsf{SemAc}(\mathbb{G})$, it is of little use for non-recursive and sticky sets of tgds. As we show in the next example, neither $\mathbb{NR}$ nor $\mathbb{S}$ have acyclicity-preserving chase:

*Example 2.* Consider the acyclic CQ and the tgd

$$q = \exists \bar{x}\big(P(x_1) \wedge \ldots \wedge P(x_n)\big) \quad \tau = P(x), P(y) \to R(x, y),$$

where $\{\tau\}$ is both non-recursive and sticky, but not guarded. In $chase(q, \{\tau\})$ the predicate $R$ holds all the possible pairs that can be formed using the terms $x_1, \ldots, x_n$. Thus, in the Gaifman graph of $chase(q, \{\tau\})$ we have an $n$-clique, which means that is highly cyclic. Notice that our example illustrates that also other favorable properties of the CQ are destroyed after chasing with non-recursive and sticky sets of tgds, namely bounded (hyper)tree width.[4] ∎

In view of the fact that the methods devised in Section 4 cannot be used for non-recursive and sticky sets of tgds, new techniques must be developed. Interestingly, $\mathbb{NR}$ and $\mathbb{S}$ share an important property, which turned out to be very useful for semantic acyclicity: *UCQ rewritability*. Recall that a *union of conjunctive queries (UCQ)* is an expression of the form $Q(\bar{x}) = \bigvee_{1 \leq i \leq n} q_i(\bar{x})$, where each $q_i$ is a CQ over the same schema $\sigma$. The evaluation of $Q$ over an instance $I$, denoted $Q(I)$, is defined as $\bigcup_{1 \leq i \leq n} q_i(I)$. The formal definition of UCQ rewritability follows:

*Definition 2.* (**UCQ Rewritability**) A class $\mathbb{C}$ of sets of tgds is *UCQ rewritable* if, for every CQ $q$, and $\Sigma \in \mathbb{C}$, we can construct a UCQ $Q$ such that: For every CQ $q'(\bar{x})$, $q' \subseteq_\Sigma q$ iff $c(\bar{x}) \in Q(D_{q'})$, with $D_{q'}$ be the database obtained from $q'$ after replacing each variable $x$ with $c(x)$. ∎

In other words, UCQ rewritability suggests that query containment can be reduced to the problem of UCQ evaluation. It is important to say that this reduction depends only on the right-hand side CQ and the set of tgds, but not on the left-hand side query. This is crucial for establishing the desirable small query property whenever we focus on sets of tgds that belong to a UCQ rewritable class. At this point, let us clarify that the class of guarded sets of tgds is not UCQ rewritable, which justifies our choice of a different semantic property, that is, acyclicity-preserving chase, for its study.

Let us now show the desirable small query property. For each UCQ rewritable class $\mathbb{C}$ of sets of tgds, there exists a computable function $f_\mathbb{C}(\cdot, \cdot)$ from the set of pairs consisting of a CQ and a set of tgds in $\mathbb{C}$ to positive integers such that the following holds: For every CQ $q$, set $\Sigma \in \mathbb{C}$, and UCQ rewriting $Q$ of $q$ and $\Sigma$, the *height* of $Q$, that is, the maximal size of its disjuncts, is at most $f_\mathbb{C}(q, \Sigma)$. The existence of the function $f_\mathbb{C}$ follows by the definition of UCQ rewritability. Then, we show the following:

PROPOSITION 15. *Let $\mathbb{C}$ be a UCQ rewritable class, $\Sigma \in \mathbb{C}$ a finite set of tgds, and $q$ a CQ. If $q$ is semantically acyclic under $\Sigma$, then there exists an acyclic CQ $q'$, where $|q'| \leq 2 \cdot f_\mathbb{C}(q, \Sigma)$, such that $q \equiv_\Sigma q'$.*

---

[4]Notice that guarded sets of tgds over predicates of bounded arity preserve the bounded hyper(tree) width of the query.

PROOF. Since $q$ is semantically acyclic under $\Sigma$, there exists an acyclic CQ $q''(\bar{x})$ such that $q \equiv_\Sigma q''$. As $\mathbb{C}$ is UCQ rewritable, there exists a UCQ $Q$ such that $c(\bar{x}) \in Q(D_{q''})$, which implies that there exists a CQ $q_r$ (one of the disjuncts of $Q$) such that $c(\bar{x}) \in q_r(D_{q''})$. Clearly, $|q_r| \leq f_\mathbb{C}(q, \Sigma)$. But $D_{q''}$ is acyclic, and thus Lemma 9 implies the existence of an acyclic CQ $q'$, where $q' \subseteq q_r$ and $|q'| \leq 2 \cdot f_\mathbb{C}(q, \Sigma)$, such that $c(\bar{x}) \in q'(D_{q''})$. The latter implies that $q'' \subseteq q'$. By hypothesis, $q \subseteq_\Sigma q''$, and hence $q \subseteq_\Sigma q'$. For the other direction, we first show that $q_r \subseteq_\Sigma q$ (otherwise, $Q$ is not a UCQ rewriting). Since $q' \subseteq q_r$, we get that $q' \subseteq_\Sigma q$. We conclude that $q \equiv_\Sigma q'$, and the claim follows. $\qquad\square$

It is clear that Proposition 15 provides a decision procedure for $\mathsf{SemAc}(\mathbb{C})$ whenever $\mathbb{C}$ is UCQ rewritable, and $\mathsf{Cont}(\mathbb{C})$ is decidable. Given a CQ $q$, and a finite set $\Sigma \in \mathbb{C}$:

1. Guess an acyclic CQ $q'$ of size at most $2 \cdot f_\mathbb{C}(q, \Sigma)$; and

2. Verify that $q \subseteq_\Sigma q'$ and $q' \subseteq_\Sigma q$.

The next result follows:

THEOREM 16. *Consider a class $\mathbb{C}$ of sets of tgds that is UCQ rewritable. If the problem $\mathsf{Cont}(\mathbb{C})$ is decidable, then $\mathsf{SemAc}(\mathbb{C})$ is also decidable.*

## 5.1 Non-Recursiveness

As already said, the key property of $\mathbb{NR}$ that we are going to exploit for solving $\mathsf{SemAc}(\mathbb{NR})$ is UCQ rewritability. For a CQ $q$ and a set $\Sigma$ of tgds, let $p_{q,\Sigma}$ and $a_{q,\Sigma}$ be the number of predicates in $q$ and $\Sigma$, and the maximum arity over all those predicates, respectively. The next result is implicit in [20]:[5]

PROPOSITION 17. $\mathbb{NR}$ *is UCQ rewritable. Furthermore,* $f_{\mathbb{NR}}(q, \Sigma) = p_{q,\Sigma} \cdot (a_{q,\Sigma} \cdot |q| + 1)^{a_{q,\Sigma}}$.

The above result, combined with Theorem 16, implies the decidability of $\mathsf{SemAc}(\mathbb{NR})$. For the exact complexity of the problem, we simply need to analyze the complexity of the non-deterministic algorithm underlying Theorem 16. Observe that when the arity of the schema is fixed the function $f_{\mathbb{NR}}$ is polynomial, and therefore Proposition 17 guarantees the existence of a polynomially sized acyclic CQ. In this case, by exploiting Proposition 3, it is easy to show that $\mathsf{SemAc}(\mathbb{NR})$ is in NExpTime, and in NP if the schema is fixed. However, things are a bit cryptic when the arity of the schema is not fixed. In this case, $f_{\mathbb{NR}}$ is exponential, and thus we have to guess an acyclic CQ of exponential size. But now the fact that $\mathsf{Cont}(\mathbb{NR})$ is in NExpTime (by Proposition 3) alone is not enough to conclude that $\mathsf{SemAc}(\mathbb{NR})$ is also in NExpTime. We need to understand better the complexity of the query containment algorithm for $\mathbb{NR}$.

Recall that given two CQs $q(\bar{x}), q'(\bar{x})$, and a finite set $\Sigma \in \mathbb{NR}$, by Lemma 1, $q \subseteq_\Sigma q'$ iff $c(\bar{x}) \in q'(chase(q, \Sigma))$. By exploiting non-recursiveness, it can be shown that if $c(\bar{x}) \in q'(chase(q, \Sigma))$, then there exists a chase sequence

$$q = I_0 \xrightarrow{\tau_0, \bar{c}_0} I_1 \xrightarrow{\tau_1, \bar{c}_1} I_2 \dots I_{n-1} \xrightarrow{\tau_{n-1}, \bar{c}_{n-1}} I_n$$

of $q$ and $\Sigma$, where $n = |q'| \cdot (b_\Sigma)^{\mathcal{O}(p_{q',\Sigma})}$, with $b_\Sigma$ be the maximum number of atoms in the body of a tgd of $\Sigma$, such that $c(\bar{x}) \in q'(I_n)$. The query containment algorithm for $\mathbb{NR}$ simply guesses such a chase sequence of $q$ and $\Sigma$, and checks whether $c(\bar{x}) \in q'(I_n)$.

---

[5]The work [20] does not consider $\mathbb{NR}$. However, the rewriting algorithm in that paper works also for non-recursive sets of tgds.

Since $n$ is exponential, this algorithm runs in non-deterministic exponential time. Now, recall that for $\mathsf{SemAc}(\mathbb{NR})$ we need to perform two containment checks where either the left-hand side or the right-hand side query is of exponential size. But in both cases the containment algorithm for $\mathbb{NR}$ runs in non-deterministic exponential time, and hence $\mathsf{SemAc}(\mathbb{NR})$ is in NExpTime. The lower bounds are inherited from $\mathsf{AcBoolCont}(\mathbb{NR})$ since $\mathbb{NR}$ is closed under connecting (see Proposition 13). Then:

THEOREM 18. $\mathsf{SemAc}(\mathbb{NR})$ *is complete for* NExpTime, *even if the arity of the schema is fixed. It becomes* NP-*complete if the schema is fixed.*

## 5.2 Stickiness

We now focus on sticky sets of tgds. As for $\mathbb{NR}$, the key property of $\mathbb{S}$ that we are going to use is UCQ rewritability. The next result has been explicitly shown in [20]:

PROPOSITION 19. $\mathbb{S}$ *is UCQ rewritable. Furthermore,* $f_\mathbb{S}(q, \Sigma) = p_{q,\Sigma} \cdot (a_{q,\Sigma} \cdot |q| + 1)^{a_{q,\Sigma}}$.

The above result, combined with Theorem 16, implies the decidability of $\mathsf{SemAc}(\mathbb{S})$. Moreover, Proposition 19 allows us to establish an optimal upper bound when the arity of the schema is fixed since in this case the function $f_\mathbb{S}$ is polynomial. In fact, we show that $\mathsf{SemAc}(\mathbb{S})$ is NP-complete when the arity of the schema is fixed. The NP-hardness is inherited from $\mathsf{AcBoolCont}(\mathbb{S})$ since $\mathbb{S}$ is closed under connecting (see Proposition 13). Now, when the arity of the schema is not fixed the picture is still foggy. In this case, the function $f_\mathbb{S}$ is exponential, and thus by following the usual guess and check approach we get that $\mathsf{SemAc}(\mathbb{S})$ is in NExpTime, while Proposition 13 implies an ExpTime lower bound. To sum up, our generic machinery based on UCQ rewritability shows that:

THEOREM 20. $\mathsf{SemAc}(\mathbb{S})$ *is in* NExpTime *and hard for* ExpTime. *It becomes* NP-*complete if the arity is fixed.*

An interesting question that comes up is whether for sticky sets of tgds a stronger small query property than Proposition 15 can be established, which guarantees the existence of a polynomially sized equivalent acyclic CQ. It is clear that such a result would allow us to establish an ExpTime upper bound for $\mathsf{SemAc}(\mathbb{S})$. At this point, one might be tempted to think that this can be achieved by showing that the function $f_\mathbb{S}$ is actually polynomial even if the arity of the schema is not fixed. The next example shows that this is not the case. We can construct a sticky set $\Sigma$ of tgds and a CQ $q$ such that, for every UCQ rewriting $Q$ for $q$ and $\Sigma$, the height of $Q$ is exponential in the arity.

*Example 3.* Let $\Sigma$ be the sticky set of tgds given below; we write $\bar{x}_i^j$ for the tuple of variables $x_i, x_{i+1}, \dots, x_j$:

$$\big\{ P_i(\bar{x}_1^{i-1}, Z, \bar{x}_{i+1}^n, Z, O), P_i(\bar{x}_1^{i-1}, O, \bar{x}_{i+1}^n, Z, O) \rightarrow$$
$$P_{i-1}(\bar{x}_1^{i-1}, Z, \bar{x}_{i+1}^n, Z, O) \big\}_{i \in \{1,\dots,n\}}.$$

Consider also the Boolean CQ

$$P_0(0, \dots, 0, 0, 1).$$

It can be shown that, for every UCQ rewriting $Q$ for $q$ and $\Sigma$, the disjunct of $Q$ that mentions only the predicate $P_n$ contains exactly $2^n$ atoms. Therefore, there is no UCQ rewriting for $q$ and $\Sigma$ of polynomial height, which in turn implies that $f_\mathbb{S}$ cannot be polynomial in the arity of the schema. $\qquad\blacksquare$

The above discussion reveals the need to identify a more refined property of stickiness than UCQ rewritability, which will allow us to close the complexity of $\mathsf{SemAc}(\mathbb{S})$ when the arity is not fixed. This is left as an interesting open problem.
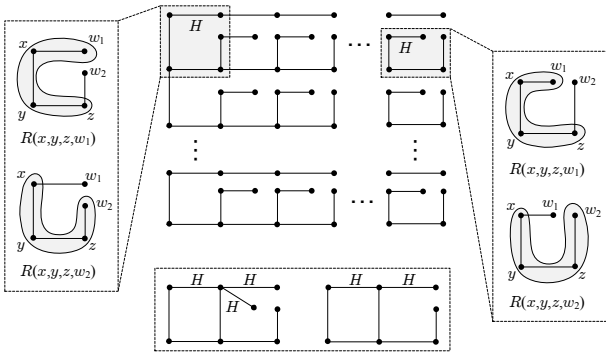
**Figure 4: From a "tree" to a grid via key dependencies.**

# 6. SEMANTIC ACYCLICITY WITH EGDS

Up to now, we have considered classes of constraints that are based on tgds. However, semantic acyclicity can be naturally defined for classes of egds. Hence, one may wonder whether the techniques developed in the previous sections can be applied for egd-based classes of constraints. Unfortunately, the situation changes dramatically even for the simplest subclass of egds, i.e., keys.

## 6.1 Peculiarity of Keys

We show that the techniques developed in the previous sections for tgds cannot be applied for showing the decidability of semantic acyclicity under keys, and thus under egds. Although the notions of acyclicity-preserving chase (Definition 1) and UCQ rewritability (Definition 2) can be naturally defined for egds, are of little use even if we focus on keys.

**Acyclicity-preserving chase.** It is easy to show via a simple example that keys over binary and ternary predicates do not enjoy the acyclicity-preserving chase property:

*Example 4.* Let $q$ be the acyclic query

$$R(x, y) \land S(x, y, z) \land S(x, z, w) \land S(x, w, v) \land R(x, v).$$

After applying on $q$ the key $R(x, y), R(x, z) \rightarrow y = z$, which simply states that the first attribute of the binary predicate $R$ is the key, we obtain the query

$$R(x, y) \land S(x, y, z) \land S(x, z, w) \land S(x, w, y),$$

which is clearly cyclic. ∎

With the aim of emphasizing the peculiarity of keys, we give a more involved example, which shows that a tree-like query can be transformed via two keys into a highly cyclic query that contains a grid. Interestingly, this shows that also other desirable properties, in particular bounded (hyper)tree width, are destroyed when we chase a query using keys.

*Example 5.* Consider the CQ $q$ depicted in Figure 4 (ignoring for the moment the dashed boxes). Although seemingly $q$ contains an $n \times n$ grid, it can be verified that the grid-like structure in the figure is actually a tree. In addition, $q$ contains atoms of the form $R(x, y, z, w)$ as explained in the figure. More precisely, for each of the open squares occurring in the first column (e.g., the upper-left shaded square), we have the two atoms $R(x, y, z, w_1)$ and $R(x, y, z, w_2)$ represented by the two hyperedges on the left. Moreover, for each of the internal open squares and the open squares occurring in the last column (e.g., the upper-right shaded square), we have the two atoms $R(x, y, z, w_1)$ and $R(x, y, z, w_2)$

represented by the two hyperedges on the right. Observe that $q$ is an acyclic query. Consider now the set $\Sigma$ of keys:

$$\epsilon_1 \;=\; R(x, y, z, w), R(x, y, z, w') \rightarrow w = w'$$
$$\epsilon_2 \;=\; H(x, y), H(x, z) \rightarrow y = z.$$

Notice that $H(\cdot, \cdot)$ stores the horizontal edges. It is not difficult to see that $chase(q, \Sigma)$ contains an $n \times n$ grid. Roughly, as described at the bottom of Figure 4, by first applying $\epsilon_1$ we close the open squares of the first column, while the open squares of the second column have now the same shape as the ones of the first column, but with a dangling $H$-edge. Then, by applying $\epsilon_2$, the two $H$-edges collapse into a single edge, and we obtain open squares that have exactly the same shape as those of the first column. After finitely many chase steps, all the squares are closed, and thus $chase(q, \Sigma)$ indeed contains an $n \times n$ grid. Therefore, although the query $q$ is acyclic, $chase(q, \Sigma)$ is far from being acyclic. Observe also that the (hyper)tree width of $chase(q, \Sigma)$ depends on $n$, while $q$ has (hyper)tree width 3. ∎

**UCQ rewritability.** It is not hard to show that keys are not UCQ rewritable. This is not surprising due to the transitive nature of equality. Intuitively, the UCQ rewritability of keys implies that a first-order (FO) query can encode the fact that the equality relation is transitive. However, it is well-known that this is not possible due to the inability of FO queries to express recursion.

## 6.2 Keys over Constrained Signatures

Despite the peculiar nature of keys as discussed above, we can establish a positive result regarding semantic acyclicity under keys, providing that only unary and binary predicates can be used. This is done by exploiting the following generic result, which is actually the version of Theorem 10 for egd-based classes:

THEOREM 21. *Consider a class $\mathbb{C}$ of sets of egds. If $\mathbb{C}$ has acyclicity-preserving chase, then* SemAc($\mathbb{C}$) *is* NP-*complete, even if we allow only unary and binary predicates.*

The proof of the above result is along the lines of the proof for Theorem 10, and exploits the fact that the containment problem under egds is feasible in non-deterministic polynomial time (this can be shown by using Lemma 1). The lower bound follows from [14], which shows that the problem of checking whether a Boolean CQ over a single binary relation is equivalent to an acyclic one is NP-hard. We now show the following positive result for the class of keys over unary and binary predicates, denoted $\mathbb{K}_2$:

PROPOSITION 22. $\mathbb{K}_2$ *has acyclicity-preserving chase.*

Notice that the above result is not in a conflict with Examples 4 and 5, since both examples use predicates of arity greater than two. It is now straightforward to see that:

THEOREM 23. SemAc($\mathbb{K}_2$) *is* NP-*complete.*

Interestingly, Theorem 23 can be extended to *unary functional dependencies* (over unconstrained signatures), that is, FDs of the form $R : A \rightarrow B$, where $R$ is a relational symbol of arity $n > 0$ and the cardinality of $A$ is one. This result has been established independently by Figueira [17]. Let us recall that egds ensure the finite controllability of CQ containment. Consequently, Theorem 23 holds even for FinSemAc, which takes as input a CQ $q$ and a set $\Sigma$ of egds, and asks for the existence of an acyclic CQ $q'$ such that $q$ and $q'$ are equivalent over all finite databases that satisfy $\Sigma$.

# 7. EVALUATION OF SEMANTICALLY ACYCLIC QUERIES

As it has been noted in different scenarios in the absence of constraints, semantic acyclicity has a positive impact on query evaluation [4, 5, 6]. We observe here that such good behavior extends to the notion of semantic acyclicity for CQs under the decidable classes of constraints we identified in the previous sections. In particular, evaluation of semantically acyclic CQs under constraints in such classes is a *fixed-parameter tractable* (fpt) problem, assuming the parameter to be $|q| + |\Sigma|$. (Here, $|q|$ and $|\Sigma|$ represent the size of reasonable encodings of $q$ and $\Sigma$, respectively). Recall that this means that the problem can be solved in time $O(|D|^c \cdot f(|q| + |\Sigma|))$, for $c \geq 1$ and $f$ a computable function.

Let $\mathbb{C}$ be a class of sets of tgds. We define $\mathsf{SemAcEval}(\mathbb{C})$ to be the following problem: The input consists of a set of constraints $\Sigma$ in $\mathbb{C}$, a semantically acyclic CQ $q$ under $\Sigma$, a database $D$ such that $D \models \Sigma$, and a tuple $\bar{t}$ of elements in $D$. We ask whether $\bar{t} \in q(D)$.

PROPOSITION 24. $\mathsf{SemAcEval}(\mathbb{C})$ *can be solved in time*

$$O\left(|D| \cdot 2^{2^{O(|q|+|\Sigma|)}}\right),$$

*where* $\mathbb{C} \in \{\mathbb{G}, \mathbb{NR}, \mathbb{S}\}$.

PROOF. Our results state that for $\mathbb{C} \in \{\mathbb{G}, \mathbb{NR}, \mathbb{S}\}$, checking if a CQ $q$ is semantically acyclic under $\mathbb{C}$ can be done in double-exponential time. More importantly, in case that $q$ is in fact semantically acyclic under $\mathbb{C}$ our proof techniques yield an equivalent acyclic CQ $q'$ of at most exponential size in $|q| + |\Sigma|$. We then compute and evaluate such a query $q'$ on $D$, and return $q(D) = q'(D)$. Clearly, this can be done in time

$$O\left(2^{2^{O(|q|+|\Sigma|)}}\right) + O\left(|D| \cdot 2^{O(|q|+|\Sigma|)}\right).$$

The running time of this algorithm is dominated by

$$O\left(|D| \cdot 2^{2^{O(|q|+|\Sigma|)}}\right)$$

and the claim follows. □

This is an improvement over general CQ evaluation for which no fpt algorithm is believed to exist [26]. It is worth remarking, nonetheless, that $\mathsf{SemAcEval}(\mathbb{C})$ corresponds to a *promise version* of the evaluation problem, where the property that defines the class is EXPTIME-hard for all the $\mathbb{C}$'s studied in Proposition 24.

The above algorithm computes an equivalent acyclic CQ $q'$ for a semantically acyclic CQ $q$ under a set of constraints in $\mathbb{C}$. This might take double-exponential time. Surprisingly, computing such $q'$ is not always needed at the moment of evaluating semantically acyclic CQs under constraints. In particular, this holds for the sets of guarded tgds. In fact, in such case evaluating a semantically acylic CQ $q$ under $\Sigma$ over a database $D$ that satisfies $\Sigma$ amounts to checking a polynomial time property over $q$ and $D$. It follows, in addition, that the evaluation problem for semantically acyclic CQs under guarded tgds is tractable:

THEOREM 25. $\mathsf{SemAcEval}(\mathbb{G})$ *is in polynomial time.*

The idea behind the proof of the above theorem is as follows. When $q$ is a semantically acyclic CQ in the absence of constraints, evaluating $q$ on $D$ amounts to checking the existence of a winning strategy for the duplicator in a particular version of the pebble game, known as the *existential 1-cover game*, on $q$ and $D$ [13]. We denote this by $q \equiv_{\exists 1c} D$. The existence of such winning

strategy can be checked in polynomial time [13]. Now, when $q$ is semantically acyclic under an arbitrary set $\Sigma$ of tgds or egds, we show that evaluating $q$ on $D$ amounts to checking whether $chase(q, \Sigma) \equiv_{\exists 1c} D$. When $\Sigma$ is a set of guarded tgds, we prove in addition that $chase(q, \Sigma) \equiv_{\exists 1c} D$ iff $q \equiv_{\exists 1c} D$. Thus, $\mathsf{SemAcEval}(\mathbb{G})$ is tractable since checking $q \equiv_{\exists 1c} D$ is tractable.

The fact that the evaluation of $q$ on $D$ boils down to checking whether $chase(q, \Sigma) \equiv_{\exists 1c} D$, when $q$ is semantically acyclic under $\Sigma$, also yields tractability for $\mathsf{SemAcEval}(\mathbb{C})$, where $\mathbb{C}$ is any class of sets of egds for which the chase can be computed in polynomial time. This includes the central class of FDs. Notice, however, that we do not know whether SemAc under FDs is decidable.

# 8. FURTHER ADVANCEMENTS

In this section we informally discuss the fact that our previous results on semantic acyclicity under tgds and CQs can be extended to UCQs. Moreover, we show that our techniques establish the existence of maximally contained acyclic queries.

## 8.1 Unions of Conjunctive Queries

It is reasonable to consider a more *liberal* version of semantic acyclicity under tgds based on UCQs. In such case we are given a UCQ $Q$ and a finite set $\Sigma$ of tgds, and the question is whether there is a union $Q'$ of acyclic CQs that is equivalent to $Q$ under $\Sigma$. It can be shown that all the complexity results on semantic acyclicity under tgds presented above continue to hold even when the input query is a UCQ. This is done by extending the small query properties established for CQs (Propositions 8 and 15) to UCQs.

Consider a finite set $\Sigma$ of tgds (that falls in one of the tgd-based classes considered above), and a UCQ $Q$. If $Q$ is semantically acyclic under $\Sigma$, then one of the following holds: (i) for each disjunct $q$ of $Q$, there exists an acyclic CQ $q'$ of bounded size (the exact size of $q'$ depends on the class of $\Sigma$) such that $q \equiv_\Sigma q'$, or (ii) $q$ is redundant in $Q$, i.e., there exists a disjunct $q'$ of $Q$ such that $q \subseteq_\Sigma q'$. Having this property in place, we can then design a nondeterministic algorithm for semantic acyclicity, which provides the desired upper bounds. Roughly, for each disjunct $q$ of $Q$, this algorithm guesses whether there exists an acyclic CQ $q'$ of bounded size such that $q \equiv_\Sigma q'$, or $q$ is redundant in $Q$. The desired lower bounds are inherited from semantic acyclicity in the case of CQs.

## 8.2 Query Approximations

Let $\mathbb{C}$ be any of the decidable classes of finite sets of tgds we study in this paper (i.e., $\mathbb{G}$, $\mathbb{NR}$, or $\mathbb{S}$). Then, for any CQ $q$ without constants[6] and set $\Sigma$ of constraints in $\mathbb{C}$, our techniques yield a *maximally contained* acyclic CQ $q'$ under $\Sigma$. This means that $q' \subseteq_\Sigma q$ and there is no acyclic CQ $q''$ such that $q'' \subseteq_\Sigma q$ and $q' \subsetneq_\Sigma q''$. Following the recent database literature, such $q'$ corresponds to an *acyclic CQ approximation of $q$ under $\Sigma$* [4, 5, 6]. Notice that when $q$ is semantically acyclic under $\Sigma$, this acyclic approximation $q'$ is in fact equivalent to $q$ under $\Sigma$. Computing and evaluating an acyclic CQ approximation for $q$ might help finding "quick" (i.e., fixed-parameter tractable) answers to it when exact evaluation is infeasible.

The way in which we obtain approximations is by slightly reformulating the small query properties established in the paper (Propositions 8 and 15). Instead of dealing with semantically acyclic CQs only, we are now given an arbitrary CQ $q$. In all cases the reformulation expresses the following: For every acyclic CQ $q'$ such that $q' \subseteq_\Sigma q$, there is an acyclic CQ $q''$ of the appropriate size $f(q, \Sigma)$

---

[6]Approximations for CQs with constants are not well-understood, even in the absence of constraints [4].

such that $q' \subseteq_\Sigma q'' \subseteq_\Sigma q$. It is easy to prove that for each CQ $q$ there exists at least one acyclic CQ $q'$ such that $q' \subseteq_\Sigma q$; take a single variable $x$ and add a tuple $R(x, \ldots, x)$ for each symbol $R$. It follows then that an acyclic CQ approximation of $q$ under $\Sigma$ can always be found among the set $\mathcal{A}(q)$ of acyclic CQs $q'$ of size at most $f(q, \Sigma)$ such that $q' \subseteq_\Sigma q$. In fact, the acyclic CQ approximations of $q$ under $\Sigma$ are the maximal elements of $\mathcal{A}(q)$ under $\subseteq_\Sigma$.

## 9. CONCLUSIONS

We have concentrated on the problem of semantic acyclicity for CQs in the presence of database constraints; in fact, tgds or egds. Surprisingly, we have shown that there are cases such as the class of full tgds, where containment is decidable, while semantic acyclicity is undecidable. We have then focussed on the main classes of tgds for which CQ containment is decidable, and do not subsume full tgds, i.e., guarded, non-recursive and sticky tgds. For these classes we have shown that semantic acyclicity is decidable, and obtained several complexity results. We have also shown that semantic acyclicity is NP-complete if we focus on keys over unary and binary predicates. Finally, we have considered the problem of evaluating a semantically acyclic CQ over a database that satisfies certain constraints, and shown that for guarded tgds and FDs is tractable. Here are some interesting open problems that we are planning to investigate: (i) The complexity of semantic acyclicity under sticky sets of tgds is still unknown; (ii) We do not know whether semantic acyclicity under keys over unconstrained signatures is decidable; and (iii) We do not know the complexity of evaluating semantically acyclic queries under $\mathbb{NR}$, $\mathbb{S}$ and egds.

## 10. REFERENCES

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

[2] J.-F. Baget, M.-L. Mugnier, S. Rudolph, and M. Thomazo. Walking the complexity lines for generalized guarded existential rules. In *IJCAI*, pages 712–717, 2011.

[3] V. Bárány, G. Gottlob, and M. Otto. Querying the guarded fragment. *Logical Methods in Computer Science*, 10(2), 2014.

[4] P. Barceló, L. Libkin, and M. Romero. Efficient approximations of conjunctive queries. *SIAM J. Comput.*, 43(3):1085–1130, 2014.

[5] P. Barceló, R. Pichler, and S. Skritek. Efficient evaluation and approximation of well-designed pattern trees. In *PODS*, pages 131–144, 2015.

[6] P. Barceló, M. Romero, and M. Y. Vardi. Semantic acyclicity on graph databases. In *PODS*, pages 237–248, 2013.

[7] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *ICALP*, pages 73–85, 1981.

[8] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

[9] A. Calì, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

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

[11] D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Trans. Comput. Log.*, 9(3), 2008.

[12] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

[13] H. Chen and V. Dalmau. Beyond hypertree width: Decomposition methods without decompositions. In *CP*, pages 167–181, 2005.

[14] V. Dalmau, P. G. Kolaitis, and M. Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP*, pages 310–326, 2002.

[15] R. Fagin. A normal form for relational databases that is based on domians and keys. *ACM Trans. Database Syst.*, 6(3):387–415, 1981.

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

[17] D. Figueira. Semantically acyclic conjunctive queries under functional dependencies. In *LICS*, 2016. To appear.

[18] T. Gogacz and J. Marcinkowski. Converging to the chase - A tool for finite controllability. In *LICS*, pages 540–549, 2013.

[19] G. Gottlob, G. Greco, and B. Marnette. Hyperconsistency width for constraint satisfaction: Algorithms and complexity results. In *Graph Theory, Computational Intelligence and Thought*, pages 87–99, 2009.

[20] G. Gottlob, G. Orsi, and A. Pieris. Query rewriting and optimization for ontological databases. *ACM Trans. Database Syst.*, 2014.

[21] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.

[22] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

[23] M. Krötzsch and S. Rudolph. Extending decidable existential rules by joining acyclicity and guardedness. In *IJCAI*, pages 963–968, 2011.

[24] T. Lukasiewicz, M. V. Martinez, A. Pieris, and G. I. Simari. From classical to consistent query answering under existential rules. In *AAAI*, pages 1546–1552, 2015.

[25] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[26] C. H. Papadimitriou and M. Yannakakis. On the complexity of database queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.

[27] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.