Semantic Acyclicity Under Constraints

Citation for published version:

Digital Object Identifier (DOI):
10.1145/2902251.2902302

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
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 oftgds 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. 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 conditions, 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:

<table>
<thead>
<tr>
<th>PROBLEM:</th>
<th>SEMANTIC ACYCLICITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT:</td>
<td>A CQ ( q ) and a finite set ( \Sigma ) of constraints.</td>
</tr>
<tr>
<td>QUESTION:</td>
<td>Is there an acyclic CQ ( q' ) s.t. ( q \equiv \Sigma q' )?</td>
</tr>
</tbody>
</table>

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 \( \forall x \forall y (\phi(x, y) \rightarrow \exists z \psi(x, z)) \), 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 \( \forall x (\phi(x) \rightarrow y = z) \), where \( \phi \) is a conjunction of atoms and \( y, z \) are variables in \( x \). 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 if and only if 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 if and only if 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 \( \text{Interest} \) contains pairs \((c, s)\) such that customer \( c \) has declared interest in style \( s \). The relation \( \text{Class} \) contains pairs \((r, s)\) such that record \( r \) is of style \( s \). Finally, the relation \( \text{Owns} \) contains a pair \((c, r)\) when customer \( c \) owns record \( r \).

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

\[
\exists z (\exists \text{Interest}(x, z) \land \exists \text{Class}(y, z) \land \exists \text{Owns}(x, y)).
\]

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 tgdb:

\[
\tau = \exists \text{Interest}(x, z) \land \exists \text{Class}(y, z) \rightarrow \exists \text{Owns}(x, y).
\]

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

\[
\exists z (\exists \text{Interest}(x, z) \land \exists \text{Class}(y, z)).
\]

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. 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 as its associated CQ containment problem: \( \text{2ExpTime-complete} \), and NP-complete for a fixed schema.

- The second criterion is **rewritability by unions of CQs (UCQs)**. Intuitively, a class \( \mathcal{C} \) of sets of tgds has this property if the CQ containment problem under a set in \( \mathcal{C} \) 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: \( \text{NExpTime-complete} \), and NP-complete if the schema is fixed. In the second case, we get a \( \text{NExpTime} \) upper bound and an \( \text{ExpTime} \) 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

---

1Modulo 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 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 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 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.

**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.
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:

$$I_0 \xrightarrow{\tau_0} I_1 \xrightarrow{\tau_1} I_2 \ldots$$

of chase steps such that: (1) $I_0 = I$; (2) For each $i \geq 0$, $\tau_i$ is a tgd in $\Sigma$; and (3) $\bigcup_{i \geq 0} I_i = \Sigma$. 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 $\text{chase}(I, \Sigma)$ the result of an arbitrary chase sequence for $I$ under $\Sigma$. Further, for a CQ $q = \exists \bar{y}(R_1(v_1) \land \ldots \land R_m(v_m))$, we denote by $\text{chase}(q, \Sigma)$ the result of the chase sequence for the database $\{R_1(v'_1), \ldots, R_m(v'_m)\}$ 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 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 $I$ 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 $\text{chase}(I, \Sigma)$. Further, for a CQ $q = \exists \bar{y}(R_1(v_1) \land \ldots \land R_m(v_m))$, we denote by $\text{chase}(q, \Sigma)$ the result of a chase sequence for the database $\{R_1(v'_1), \ldots, R_m(v'_m)\}$ 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 character-

- **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). 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.

  - **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] $\text{Cont}(\mathbb{NR})$ is complete for $\text{NP}$-complete, even if the arity of the schema is fixed. It becomes $\text{NP}$-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

3In fact, these restrictions are designed to obtain decidable query answering under tgds. However, this problem is equivalent to query containment under tgds (Lemma 1).
Each one of the previous classes has an associated weak version, called weakly-sticky.

A key property of all these classes is that they extend the class of full tgds, by giving a solution to SemAc(C) we immediately obtain a solution for FinSemAc(C).

The reason why we prefer to focus on SemAc(C), instead of FinSemAc(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. 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: \( \mathcal{C} \) is a class of sets of tgds (e.g., guarded, non-recursive, sticky, etc.).

**Problem**: \( \text{SemAc}(\mathcal{C}) \)

**Input**: A CQ \( q \) and a finite set \( \Sigma \) of tgds in \( \mathcal{C} \)

**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 SemAc(\( \mathcal{C} \)) 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 FinSemAc(\( \mathcal{C} \)), which accepts as input a CQ \( q \) and a finite set \( \Sigma \in \mathcal{C} \) 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, SemAc and 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 \( \mathcal{C} \) is one of the above syntactic classes of sets of tgds, by giving a solution to SemAc(\( \mathcal{C} \)) we immediately obtain a solution for FinSemAc(\( \mathcal{C} \)).

**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 \( \Sigma \) contains two occurrences of a marked variable. Then:

**Proposition 4.** [10] Cont(\( \mathcal{S} \)) 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.

### 3.2 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 \( q \not\subseteq_{\Sigma} q' \) iff \( q \land q' \) is semantically acyclic under \( \Sigma \).

As an immediate corollary of Proposition 5, we obtain an initial boundary for the decidability of 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 RestCont(\( \mathcal{C} \)) to be the problem of checking \( q \subseteq_{\Sigma} \{ q', \} \), given a set \( \Sigma \) of body-connected tgds in \( \mathcal{C} \) 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.** SemAc(\( \mathcal{C} \)) is undecidable for every class \( \mathcal{C} \) of tgds such that RestCont(\( \mathcal{C} \)) is undecidable.

As we shall discuss later, 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 \( \mathcal{C} \) is reducible to the containment problem for \( \mathcal{C} \). This would imply the decidability of 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 SemAc is undecidable even over the class \( \mathcal{F} \) of sets of full tgds, which ensures the decidability of CQ containment:

**Theorem 7.** The problem SemAc(\( \mathcal{F} \)) 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 \( w'_1, \ldots, w'_m \) 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 \( w_{i_1} \cdots w_{i_m} = w'_{i_1} \cdots w'_{i_m} \).
Let \( w_1, \ldots, w_n \) and \( w'_1, \ldots, w'_m \) 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 \( \{ P_a, P_b, P_y, \text{sync, start, end} \} \), where \( P_a, P_b, P_y, \text{sync} \) are binary predicates, and start and end are unary predicates, such that the PCP instance given by \( w_1, \ldots, w_n \) and \( w'_1, \ldots, w'_m \) has a solution iff there is an acyclic CQ \( q' \) such that \( q \equiv \Sigma \, q' \). In this section 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 \( 1 \leq i \leq |w| \), the prefix of \( w \) of length \( i \). In such a 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 \( \langle [w[i], w[j]] \rangle \) belongs to the interpretation of sync.

Formally, \( \Sigma \) consists of the following rules:

1. An initialization rule:
   \[
   \text{start}(x), P_y(x, y) \rightarrow \text{sync}(y, y).
   \]
   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 \( 1 \leq i \leq n \), a synchronization rule:
   \[
   \text{sync}(x, y), P_{a_i}(x, z), P_{a_j}(y, u) \rightarrow \text{sync}(z, u).
   \]
   Here, \( P_{a_i}(x, y), \) for \( w = a_1 \ldots a_t \in \{ a, b \}^+ \), denotes \( P_{a_i}(x, x_1), \ldots, P_{a_i}(x_{t-1}, y) \), 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., \( w_j \)), then \( (z, u) \) is also synchronized.

3. For each \( 1 \leq i \leq n \), a finalization rule:
   \[
   \text{start}(x), P_a(y, z), P_a(z, u), P_c(u, v), \text{end}(v), \text{sync}(y_1, y_2), P_{a_i}(y, y), P_{a_j}(y, y) \rightarrow \psi,
   \]
   where \( \psi \) is the conjunction of atoms:
   \[
   P_y(x, y), P_y(x, u), P_y(y, v), P_y(y, v), \]
   \[
   P_b(z, y), P_b(u, z), P_a(u, y), P_b(y, u),
   \]
   \[
   \text{sync}(y, y), \text{sync}(z, z), \text{sync}(y, z), \text{sync}(z, y), \text{sync}(y, u), \text{sync}(u, y), \text{sync}(z, u), \text{sync}(u, z).
   \]

This tgd enforces \( \text{chase}(q', \Sigma) \) 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 \( 1 \leq i_1, \ldots, i_m \leq n \), 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:

\[
\text{start} \rightarrow \text{end} \rightarrow \ldots \rightarrow \text{start} \rightarrow \text{end} \rightarrow \ldots
\]

This is a generalization of the one we sketch below. The proof is by induction on \( |q'| \).

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 \( \text{SmmAC} \), 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 \( \text{SmmAC} \) whenever the problem \( \text{Cont}(\Sigma) \) is decidable. This criterion guarantees that, starting from an acyclic instance, it is not possible to destroy
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 $\text{SemAc}(G)$, 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 $C$ of sets of tgds has acyclicity-preserving chase if, for every acyclic CQ $q$, set $\Sigma \in C$, 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 $\alpha_1, \ldots, \alpha_3$ are the atoms of $q$, there exists a homomorphism $\mu$ that maps $\alpha_1 \land \ldots \land \alpha_3$ 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 $\text{chase}(q'', \Sigma)$. Recall that $\Sigma$ belongs to a class that has acyclicity-preserving chase, which implies that $\text{chase}(q'', \Sigma)$ 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 $\text{chase}(q'', \Sigma)$. By Lemma 1, $q'' \subseteq q'$, and therefore $q \equiv_\Sigma q'$. We conclude that $q \equiv_\Sigma q'$, and the claim follows.

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

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 $C$ of sets of tgds that has acyclicity-preserving chase. If the problem $\text{Cont}(C)$ is decidable, then $\text{SemAc}(C)$ is also decidable.

### 4.1 Guardedness

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

**Theorem 11.** $\text{SemAc}(G)$ is complete for $2\text{ExpTime}$. It becomes $\text{ExpTime}$-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.** $G$ has acyclicity-preserving chase.

The above result, combined with Theorem 10, implies the decidability of $\text{SemAc}(G)$. However, this does not say anything about the complexity of the problem. With the aim of pinpointing the exact complexity of $\text{SemAc}(G)$, we proceed to analyze the complexity of the decision procedure underlying Theorem 10. Recall that, given a CQ $q$, and a finite set $\Sigma \in G$, 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 $\text{Cont}(G)$. Thus, Proposition 2 implies that $\text{SemAc}(G)$ is in $2\text{ExpTime}$, in $\text{ExpTime}$ 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 $\Sigma^P_2$. Observe that the oracle is called only once in order to solve $\text{Cont}(G)$, and since $\text{Cont}(G)$ 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, $\text{RestCont}(G)$ can be reduced in constant time to $\text{SemAc}(G)$. Thus, to obtain the desired lower bounds, it suffices to
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 \( \exists y (R_1(\bar{v}_1, w) \land \cdots \land R_m(\bar{v}_m, w) \land \text{aux}(w, w)) \). The application of the connecting operator on \( (q, q', \Sigma) \) returns the triple \((c(q), c(q'), c(\Sigma))\), where

- \( c(q) \) is the query \( \exists y \exists w (R_1(\bar{v}_1, w) \land \cdots \land R_m(\bar{v}_m, w) \land \text{aux}(w, w)) \), where \( w \) is a new variable not in \( q \), each \( R_i^* \) is a new predicate, and \( \text{aux} \) is a new binary predicate;
- \( c(q') \) is the query \( \exists y \exists u \exists v \exists w (R_1(\bar{v}_1, w) \land \cdots \land R_m(\bar{v}_m, w) \land \text{aux}(w, u) \land \text{aux}(u, v) \land \text{aux}(v, w)) \), where \( w, u, v \) are new variables not in \( q \); and
- \( c(\Sigma) = \{ c(\tau) \mid \tau \in \Sigma \} \), where for a tgd \( \tau = \phi(x, y) \rightarrow \exists z \psi(x, z) \), \( c(\tau) \) is the tgd \( \phi^*(x, y, w) \rightarrow \exists z \exists \psi^*(x, z, w) \), with \( \phi^*(x, y, w) \) and \( \psi^*(x, z, w) \) being the conjuncts obtained from \( \phi(x, y) \) and \( \psi(x, z) \), respectively, by replacing each atom \( R(x_1, \ldots, x_n) \) with \( R^*(x_1, \ldots, x_n, w) \), where \( w \) is a new variable not occurring in \( \tau \).

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

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

**Proposition 13.** Let \( C \) be a class of sets of tgds that is closed under connecting such that AcBoolCont(\( C \)) is hard for a complexity class \( C' \) that is closed under polynomial time reductions. Then, SemAc(\( C \)) is also \( C' \)-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 SemAc(\( G \)) stated in Theorem 11 follow from Propositions 2 and 13. Note that, although Proposition 2 refers to Cont(\( G \)), the lower bounds hold for AcBoolCont(\( G \)); 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 SemAc(\( G \)), and the fact that both linear tgds and inclusion dependencies are closed under connecting, we can show that:

**Theorem 14.** SemAc(\( C \)), for \( C \in \{ \mathbb{L}, \mathbb{I} \} \), 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 SemAc(\( G \)), it is of little use for non-recursive and sticky sets of tgds. As we show in the next example, neither \( \mathbb{N}RT \) nor \( \mathbb{S} \) have acyclicity-preserving chase:

**Example 2.** Consider the acyclic CQ and the tgd
\[
q = \exists x (P(x_1) \land \cdots \land P(x_n)) \quad \tau = P(x), P(y) \rightarrow R(x, y),
\]
where \( \{ \tau \} \) is both non-recursive and sticky, but not guarded. In \( \text{chase}(q, \{ \tau \}) \) the predicate \( R \) holds all the possible pairs that can be formed using the terms \( x_1, \ldots, x_n \). Thus, in the Graftin graph of \( \text{chase}(q, \{ \tau \}) \) we have an \( n \times n \)-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.

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, \( \mathbb{N}RT \) 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 \( C \) of sets of tgds is UCQ rewritable if, for every \( C \) set of tgds, there exists a computable function \( f_C(\cdot, \cdot) \) from the set of pairs consisting of a CQ and a set of sets of tgds to the set of pairs consisting of a UCQ and its size. Then, \( f_C \) is UCQ rewritability.

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 \( C \) of sets of tgds, there exists a computable function \( f_C(\cdot, \cdot) \) from the set of pairs consisting of a CQ and a set of sets of tgds in to positive integers such that the following holds: For each CQ \( q \) set \( \Sigma \) of \( C \) and UCQ rewriting \( Q \) of \( q \) and \( \Sigma \), the height of \( Q \) is the maximal size of its disjuncts, is at most \( f_C(q, \Sigma) \). The existence of the function \( f_C \) follows by the definition of UCQ rewritability. Then, we show the following:

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

\(^2\) Notice that guarded sets of tgds over predicates of bounded arity preserve the bounded (hyper)tree width of the query.
It is clear that Proposition 15 provides a decision procedure for SemAc(Σ) whenever Σ is UCQ rewritable, and Cont(Σ) is decidable. Given a CQ q, and a finite set Σ ∈ C:

1. Guess an acyclic CQ q′ of size at most 2 · f_c(q, Σ); and

2. Verify that q ⊆ q′ and q′ ⊆ q.

The next result follows:

**Theorem 16.** Consider a class C of sets of tgds that is UCQ rewritable. If the problem Cont(Σ) is decidable, then SemAc(Σ) is also decidable.

### 5.1 Non-Recursiveness

As already said, the key property of NR that we are going to exploit for solving SemAc(Σ) is UCQ rewritability. For a CQ q and a set Σ of tgds, let p_q,Σ; and a_q,Σ be the number of predicates in q and Σ, and the maximum arity over all those predicates, respectively. The next result is implicit in [20].

**Proposition 17.** NR is UCQ rewritable. Furthermore, f_bar(b,q,Σ) = p_q,Σ · (a_q,Σ · |q| + 1)^a_q,Σ.

The above result, combined with Theorem 16, implies the decidability of SemAc(Σ). 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_bar 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 SemAc(Σ) 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_bar is exponential, and thus we have to guess an acyclic CQ of exponential size. But now the fact that Cont(Σ) is in NEXPTIME (by Theorem 3) alone is not enough to conclude that SemAc(Σ) is also in NEXPTIME. We need to understand better the complexity of the query containment algorithm for NR.

Recall that given two CQs q(x), q′(x), and a finite set Σ ∈ NR, by Lemma 1, q ⊆ q′ iff c(x) ∈ q′(chase(q, Σ)). By exploiting non-rewrivity, it can be shown that if c(x) ∈ q′(chase(q, Σ)), then there exists a chase sequence

\[ q = I_0 \xrightarrow{\tau_0,\xi_0} I_1 \xrightarrow{\tau_1,\xi_1} I_2 \ldots \xrightarrow{\tau_n,\xi_{n-1}} I_n \]

of q and Σ, where \( n = |q'| \cdot (b_0)^{|q'|,\xi} \), with b_0 being the maximum number of atoms in the body of a tgd of Σ, such that c(x) ∈ q′(I_0).

The query containment algorithm for NR simply guesses such a chase sequence of q and Σ, and checks whether c(x) ∈ q′(I_0).

Since n is exponential, this algorithm runs in non-deterministic exponential time. Now, recall that for SemAc(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 NR runs in non-deterministic exponential time, and hence SemAc(NR) is in NEXPTIME. The lower bounds are inherited from AcBoolCont(NR) since NR is closed under connecting (see Proposition 13).

**Theorem 18.** SemAc(Σ) 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 NR, the key property of S that we are going to use is UCQ rewritability. The next result has been explicitly shown in [20]:

**Proposition 19.** S is UCQ rewritable. Furthermore, f_s(q,Σ) = p_q,Σ · (a_q,Σ · |q| + 1)^a_q,Σ.

The above result, combined with Theorem 16, implies the decidability of SemAc(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_s is polynomial. In fact, we show that SemAc(S) is NP-complete when the arity of the schema is fixed. The NP-hardness is inherited from AcBoolCont(S) since 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_s is exponential, and thus by following the usual guess and check approach we get that SemAc(S) is in NEXPTIME, while Proposition 13 implies an EXP-TIME lower bound. To sum up, our generic machinery based on UCQ rewritability shows that:

**Theorem 20.** SemAc(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 EXP-TIME upper bound for SemAc(S). At this point, one might be tempted to think that this can be achieved by showing that the function f_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 Σ of tgds and a CQ q such that, for every UCQ rewriting Q for q and Σ, the height of Q is exponential in the arity.

**Example 3.** Let Σ be the sticky set of tgds given below; we write \( x_i^k \) for the tuple of variables \( x_1, x_{i+1}, \ldots, x_j \):

\[ \{ P_1(x_1^{i-1}, Z, x_{i+1}^n, Z, O), P_1(x_1^{i-1}, O, x_{i+1}^n, Z, O) \} \}_{i \in \{1, \ldots, n\}} \]

Consider also the Boolean CQ

\[ P_b(0, 0, 0, \ldots, 0, 0, 1) \]

It can be shown that, for every UCQ rewriting Q for q and Σ, the disjunct of Q that mentions only the predicate \( P_1 \) contains exactly \( 2^n \) atoms. Therefore, there is no UCQ rewriting for q and Σ of polynomial height, which in turn implies that f_s cannot be polynomial in the arity of the schema.

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 SemAc(S) when the arity is not fixed. This is left as an interesting open problem.
Moreover, for each of the internal open squares and the open upper-left shaded square, we have the two atoms for each of the open squares occurring in the first column (e.g., the n-ary 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 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.

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 \( \mathcal{C} \) of sets of egds. If \( \mathcal{C} \) has acyclicity-preserving chase, then \( \text{SemAc}(\mathcal{C}) \) 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. \( \text{SemAc}(\mathbb{K}_2) \) 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 \( \text{FinSemAc} \), 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 $O(|D| \cdot f(|q| + |\Sigma|))$, for $c \geq 1$ and $f$ a computable function.

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

**Proposition 24.** $\text{SemAcEval}(C)$ can be solved in time

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

where $C \in \{G, N_R, S\}$.

**Proof.** Our results state that for $C \in \{G, N_R, S\}$, checking if a CQ $q$ is semantically acyclic under $C$ can be done in double-exponential time. More importantly, in case that $q$ is in fact semantically acyclic under $C$ 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 $q(D) \equiv q'(D)$. Clearly, this can be done in time

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

The running time of this algorithm is dominated by

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

and the claim follows. $\square$

This is an improvement over general CQ evaluation for which no fpt algorithm is believed to exist [26]. It is worth remarking, nonetheless, that $\text{SemAcEval}(C)$ corresponds to a promise version of the evaluation problem, where the property that defines the class is EXPTIME-hard for all the $C$’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 $C$. 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 acyclic 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.** $\text{SemAcEval}(\Sigma)$ 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 $I$-cover game, on $q$ and $D$ [13]. We denote this by $q \equiv_{31e} D$. 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 $\text{chase}(q, \Sigma) \equiv_{31e} D$. When $\Sigma$ is a set of guarded tgds, we prove in addition that $\text{chase}(q, \Sigma) \equiv_{31e} D$ if and only if $q \equiv_{31e} D$. Thus, $\text{SemAcEval}(\Sigma)$ is tractable since checking $q \equiv_{31e} D$ is tractable.

The fact that the evaluation of $q$ on $D$ boils down to checking whether $\text{chase}(q, \Sigma) \equiv_{31e} D$, when $q$ is semantically acyclic under $\Sigma$, also yields tractability for $\text{SemAcEval}(C)$, where $C$ 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 $\text{SemAc}$ under FDs is decidable.

8. FURTHER ADVANCES

In this section we informally discuss the fact that our previous results on semantic acyclicity under tgds and CQs can be extended to UCNs. 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 UCNs. In such case we are given a UCN $Q$ 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 UCN. This is done by extending the small query properties established for CQs (Propositions 8 and 15) to UCNs.

Consider a finite set $\Sigma$ of tgds (that falls in one of the tgd-based classes considered above), and a UCN $Q$. 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 $q \subseteq q'$. Having this property in place, we can then design a non-deterministic 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 $C$ be any of the decidable classes of finite sets of tgds we study in this paper (i.e., $G$, $N_R$, or $S$). Then, for any CQ $q$ without constants and set $\Sigma$ of constraints in $C$, our techniques yield a maximally contained acyclic CQ $q'$ under $\Sigma$. This means that $q' \subseteq \Sigma$ and there is no acyclic CQ $q''$ such that $q'' \subseteq \Sigma$ and $q' \subseteq q''$. 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 $q' \subseteq q$, there is an acyclic CQ $q''$ of the appropriate size $f(q, \Sigma)$

\[\text{Approximations for CQs with constants are not well-understood, even in the absence of constraints [4].}\]
such that \( q' \subseteq \Sigma \subseteq q \). It is easy to prove that for each CQ \( q \) there exists at least one acyclic CQ \( q' \) such that \( q' \subseteq q \); 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 \( q' \subseteq q \). In fact, the acyclic CQ approximations of \( q \) under \( \Sigma \) are the maximal elements of \( \mathcal{A}(q) \) under \( \subseteq \).

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 it 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 \( NR \), \( S \) and egds.

Acknowledgements: Barceló would like to thank D. Figueira, M. Romero, S. Rudolph, and N. Schweikardt for insightful discussions about the nature of semantic acyclicity under constraints. We also thank the anonymous referees for their helpful feedback. Barceló is funded by the Millenium Nucleus Center for Semantic Web Research under grant NC120004. Gottlob is supported by the EPSRC Programme Grant EP/M025268/ “VADA: Value Added Data Systems – Principles and Architecture”. Pieris is supported by the Austrian Science Fund (FWF), projects P25207-N23 and Y698, and Vienna Science and Technology Fund (WWTF), project ICT12-015.

10. REFERENCES