ABSTRACT

The chase procedure is considered as one of the most fundamental algorithmic tools in database theory. It has been successfully applied to different database problems such as data exchange, and query answering and containment under constraints, to name a few. One of the central problems regarding the chase procedure is all-instance termination, that is, given a set of tuple-generating dependencies (TGDs) (a.k.a. existential rules), decide whether the chase under that set terminates, for every input database. It is well-known that this problem is undecidable, no matter which version of the chase we consider. The crucial question that comes up is whether existing restricted classes of TGDs, proposed in different contexts such as ontological query answering, make the above problem decidable. In this work, we focus our attention on the oblivious and the semi-oblivious versions of the chase procedure, and we give a positive answer for classes of TGDs that are based on the notion of guardedness. To the best of our knowledge, this is the first work that establishes positive results about the (semi-)oblivious chase termination problem. In particular, we first concentrate on the class of linear TGDs, and we syntactically characterize, via rich- and weak-acyclicity, its fragments that guarantee the termination of the oblivious and the semi-oblivious chase, respectively. Those syntactic characterizations, apart from being interesting in their own right, allow us to pinpoint the complexity of the problem, which is \( \mathbf{PSPACE}\)-complete in general, and \( \mathbf{NL}\)-complete if we focus on predicates of bounded arity, for both the oblivious and the semi-oblivious chase. We then proceed with the more general classes of guarded and weakly-guarded TGDs. Although we do not provide syntactic characterizations for its relevant fragments, as for linear TGDs, we show that the problem under consideration remains decidable. In fact, we show that it is \( \mathbf{2EXPTIME}\)-complete in general, and \( \mathbf{EXPTIME}\)-complete if we focus on predicates of bounded arity, for both the oblivious and the semi-oblivious chase. Finally, we investigate the expressive power of the query languages obtained from our analysis, and we show that they are equally expressive with standard database query languages. Nevertheless, we have strong indications that they are more succinct.

Categories and Subject Descriptors

F.2.2 [Analysis of Algorithms and Problem Complexity]: Non-numerical Algorithms and Problems—complexity of proof procedures, computations on discrete structures; H.2.4 [Database Management]: Systems—relational databases, rule-based databases

Keywords

Chase Procedure; Termination; Tuple-Generating Dependencies; Existential Rules; Guardedness; Decidability; Complexity

1. INTRODUCTION

1.1 The Chase Procedure

The chase procedure (or simply chase) is considered as one of the most fundamental algorithmic tools in databases. It has been successfully applied to a wide range of problems: containment of queries under constraints [2], checking logical implication of constraints [5, 21], computing data exchange solutions [11], and query answering under constraints [6], to name a few. The chase procedure takes as input a database \( D \) and a set \( \Sigma \) of constraints and, if it terminates (which is not guaranteed), its result is a finite instance \( D_\Sigma \) that enjoys two crucial properties:

1. \( D_\Sigma \) is a model of \( D \) and \( \Sigma \), i.e., it contains \( D \) and satisfies the constraints of \( \Sigma \); and
2. \( D_\Sigma \) is universal, i.e., it can be homomorphically embedded into every other model of \( D \) and \( \Sigma \).

In other words, the chase is an algorithmic tool for computing universal models of \( D \) and \( \Sigma \), which can be conceived as representatives of all the other models of \( D \) and \( \Sigma \). This is precisely the reason for the ubiquity of the chase in database theory, as accurately discussed in [9]. Indeed, many key database problems, as the ones above, can be solved by simply exhibiting a universal model.

A central class of constraints, which can be treated by the chase procedure, is the class of tuple-generating dependencies (TGDs) (a.k.a. existential rules). TGDs are implications of the form

\[
\forall X \forall Y (\varphi(X, Y) \rightarrow \exists Z \psi(Y, Z)),
\]

where \( \varphi \) and \( \psi \) are conjunctions of atoms, and they essentially state that the presence of some tuples in an instance implies the existence of some other tuples in the same instance. Given a database \( D \) and a set \( \Sigma \) of TGDs, the chase adds new atoms to \( D \) (possibly involving nulls that act as witnesses for the existentially quantified variables) until the final result satisfies \( \Sigma \).

Example 1. Consider the database \( D = \{ \text{person}(\text{Bob}) \} \), and the TGD \( \forall X \exists Y \text{hasFather}(X, Y) \land \text{person}(Y) \).
which asserts that each person has a father who is also a person. The database atom \textit{triggers} the TGD, and the chase will add in \( D \) the atoms hasFather(Bob, \( z_1 \)) and person(\( z_1 \)) in order to satisfy it, where \( z_1 \) is a (labeled) null representing some unknown value. However, the new atom \textit{person}(\( z_1 \)) triggers again the TGD, and the chase is forced to add the atoms hasFather(\( z_1, z_2 \)), person(\( z_2 \)), where \( z_2 \) is a new null. The result of the chase is the instance

\[
\{ \text{person}(\text{Bob}), \text{hasFather}(\text{Bob}, \text{\( z_1 \})) \} \cup \\
\bigcup_{i>0} \{ \text{person}(\text{\( z_i \)}), \text{hasFather}(\text{\( z_i, z_{i+1} \})) \},
\]

where \( z_1, z_2, \ldots \) are nulls.

### 1.2 The Challenge of Infinity

As shown by Example 1, the chase procedure may run forever, even for extremely simple databases and constraints. In the light of this fact, there has been a long line of research on identifying syntactic properties on TGDs such that, for every input database, the termination of the chase is guaranteed. A prime example of such a property is \textit{weak-acyclicity} [11], which forms the standard language for data exchange purposes, and guarantees the termination of the semi-oblivious and restricted chase. A similar formalism, called \textit{constraints with stratified-witness}, has been proposed in [10]. Inspired by weak-acyclicity, the notion of \textit{rich-acyclicity} has been proposed in [19], which guarantees the termination of the oblivious chase. Note that the key difference between the various versions of the chase procedure is when a TGD is triggered. Many other sufficient conditions for chase termination can be found in the literature; see, e.g., [9, 16, 18, 22, 24] — this list is by no means exhaustive, and we refer to [17] for a comprehensive survey.

With so much effort spent on identifying sufficient conditions for the termination of the chase procedure, the question that comes up is whether a sufficient condition that is also \textit{necessary} exists. In other words, given a set \( \Sigma \) of TGDs, is it possible to determine whether, for every database \( D \), the chase on \( D \) and \( \Sigma \) terminates? This interesting question has been recently addressed in [12], and unfortunately the answer is negative for all the versions of the chase that are usually used in database applications, namely the oblivious, semi-oblivious, and restricted chase. The problem remains undecidable even if the database is known; this has been established in [9] for the restricted chase, and it was observed in [22] that the same proof shows undecidability also for the (semi-)oblivious chase.

### 1.3 Towards Positive Results

Although the chase termination problem is undecidable in general, the proof in [12] does not show the undecidability of the problem for TGDs that enjoy some structural conditions, which in turn guarantee favorable model-theoretic properties. Such a condition is \textit{guardedness}, a well-accepted paradigm that gives rise to robust rule-based languages [4, 6, 7] that capture important databases constraints such as inclusion dependencies, and lightweight description logics such as DL-Lite [8] and \( \mathcal{EL} \) [3]. A TGD is guarded if it has an atom in the left-hand side that contains (or guards) all the universally quantified variables. Guardedness guarantees the tree-likeness of the underlying models, and thus the decidability of central database problems such as query answering under constraints. The question that comes up is whether guardedness has the same positive impact on the chase termination problem:

**Question 1:** Given a set \( \Sigma \) of guarded TGDs, is it possible to decide whether, for every database \( D \), the chase on \( D \) and \( \Sigma \) terminates?

Of course, if the answer to the above question is positive, then the next step is to understand how complex is the problem of determining whether the chase terminates:

**Question 2:** Given a set \( \Sigma \) of guarded TGDs, what is the exact complexity of deciding whether, for every database \( D \), the chase on \( D \) and \( \Sigma \) terminates?

Our main goal in this work is to study in depth the chase termination problem for guarded TGDs, and give answers to the above fundamental questions. In fact, we focus on the (semi-)oblivious versions of the chase, and we show that deciding termination for guarded TGDs is decidable. Surprisingly, this work is to our knowledge the first one that establishes positive results for the chase termination problem. Although the (semi-)oblivious versions of the chase are considered as non-standard ones, they have certain advantages that make them as important as the restricted chase, and thus they deserve our attention. In particular, unlike the restricted chase, the application of a TGD does not require checking if the head of the TGD is already satisfied by the instance, and this guarantees technical clarity and efficiency; see [6, 22] for a discussion on the advantages of the oblivious and semi-oblivious chase.

From our analysis, it turned out that to decide the termination of the chase is inherently different from query answering under guarded TGDs. For our purposes, we had to tame the combinatorial nature of the chase procedure, and understand how different chase derivations affect each other during the construction of the chase. More precisely, we had to understand when a sequence of TGDs gives rise to a valid derivation during the construction of the chase, and whether such a derivation is infinite. These low level issues are irrelevant for query answering, and this is the reason why the chase termination problem is generally considered more challenging than query answering.

It is clear that our positive results immediately give rise to new decidable query languages that can directly exploit the chase procedure. Another goal of the present paper is to clarify the expressiveness of those languages:

**Question 3:** What is the relative expressive power of the query language obtained from the fragment of guarded TGDs that ensures the termination of the chase?

### 1.4 Impact

It is interesting to observe that key database problems may benefit from our results. Such problems include: (i) computing data exchange solutions [11], and (ii) answering conjunctive queries in the presence of guarded TGDs [6, 7].

For data exchange, it is vital to use languages that guarantee the termination of the chase, since data exchange solutions must be materialized, and thus explicitly computable. Recall that the main language for data exchange is the class of weakly-acyclic TGDs. However, there are several data exchange scenarios that can be expressed via guarded TGDs, but not via weakly-acyclic TGDs. Our results provide formal algorithmic tools for checking whether a set of guarded TGDs, which is not necessarily weakly-acyclic, is suitable for data exchange purposes.

Regarding query answering, there are decision procedures that allow us to answer conjunctive queries in the presence of guarded TGDs, even if the chase is infinite. The main idea underlying these procedures is to compute an initial finite portion of the chase, whose size depends on the TGDs and the query, and then evaluate the query over this finite instance. It is clear that, by following this approach, for queries of different size we are forced to compute the relevant part of the chase, which is an expensive task. However,
if we know that the given set of TGDs guarantees the termination of the chase, then we can simply compute once a (finite) universal model, and then evaluate queries directly on that model.

1.5 Summary of Contributions

We first concentrate, in Section 4, on linear and simple linear TGDs, two key subclasses of guarded TGDs. Linear TGDs have only one atom in the left-hand side, while simple linear TGDs do not allow the repetition of variables in the left-hand side. Despite their simplicity, the above classes are powerful enough for capturing prominent database dependencies, and in particular inclusion dependencies, as well as key description logics such as DL-Lite. We syntactically characterize the fragment of linear and simple linear TGDs that ensures the termination of the oblivious and semi-oblivious chase via rich- and weak-acyclicity, respectively. More precisely, we show that a set of simple linear TGDs ensures the termination of the oblivious (resp., semi-oblivious) chase iff it is richly-acyclic (resp., weakly-acyclic). However, for linear TGDs we need to carefully extend rich- and weak-acyclicity. After exposing the reasons why the above acyclicity notions are not powerful enough for our purposes, we introduce critical-rich-acyclicity and critical-weak-acyclicity, and we show that they characterize the fragment of linear TGDs that guarantees the termination of the oblivious and semi-oblivious chase, respectively.

The above syntactic characterizations, apart from being interesting in their own right, allow us to obtain optimal upper bounds for the chase termination problem under (simple) linear TGDs — we simply need to analyze the complexity of deciding whether a set of (simple) linear TGDs enjoys the above acyclicity-based conditions. In particular, we show that the problem for simple linear TGDs is NL-complete, even for unary and binary predicates, while for linear TGDs it is PSPACE-complete, in general, and NL-complete for predicates of bounded arity. For the hardness results, a generic technique, called the looping operator, is proposed, which allows us to obtain lower bounds for the chase termination problem in a uniform way. In fact, the goal of the looping operator is to provide a generic reduction from propositional atom entailment to the complement of chase termination.

We then proceed, in Section 5, with guarded and the more general language of weakly-guarded TGDs. Although there is no way (at least no obvious one) to syntactically characterize the fragments of (weakly-)guarded TGDs that ensure the termination of the chase, it is possible to show that the problem of recognizing the above classes is decidable, and in particular 2EXPTIME-complete, in general, and EXPTIME-complete for predicates of bounded arity. The upper bounds are obtained by exhibiting an alternating algorithm that runs in exponential space, in general, and in polynomial space in case of predicates of bounded arity. The lower bounds are obtained by reductions from the acceptance problem of alternating exponential (resp., polynomial) space clocked Turing machines, i.e., Turing machines equipped with a counter. These reductions are obtained by modifying significantly existing reductions for the problem of propositional atom entailment under (weakly-)guarded TGDs, and then exploiting the looping operator mentioned above. The complexity results in this paper are summarized in Table 1.

Finally, in Section 6, we investigate the expressive power of our languages. In particular, we show that the query language based on the fragment of (simple) linear TGDs that guarantees the termination of the chase has the same expressive power as (simple) linear UCQs (unions of conjunctive queries). Similarly, we show that the query language based on the fragment of (weakly-)guarded TGDs that guarantees the termination of the chase has the same expressive power as (weakly-)guarded Datalog. The above results show that the new languages obtained from our analysis do not provide us with more expressive power compared to the standard database query languages. Nevertheless, we have a strong indication that the fragment of (weakly-)guarded TGDs that guarantees the termination of the chase is more succinct than (weakly-)guarded Datalog.

2. PRELIMINARIES

2.1 General Definitions

We define the following pairwise disjoint sets of symbols: a set \( C \) of constants (constitute the “normal” domain of a database), a set \( N \) of (labeled) nulls (used as placeholders for unknown values, and thus can be also seen as (globally) existentially quantified variables), and a set \( V \) of (regular) variables (used in dependencies). A fixed lexicographic order is assumed on \((C \cup N)\) such that every null of \( N \) follows all constants of \( C \). We denote by \( X \) sequences (or sets, with a slight abuse of notation) of variables or constants \( X_1, \ldots, X_k \), with \( k \geq 0 \). Throughout, let \([n] = \{1, \ldots, n\}\) for any integer \( n \geq 1 \).

A (relational) schema \( \mathcal{R} \) is a (finite) set of relational symbols (or predicates), each with its associated arity. We write \( p/n \) to denote that \( p \) is an \( n \)-ary predicate. A position \( p[i] \) in \( \mathcal{R} \) is identified by a predicate \( p \in \mathcal{R} \) and its \( i \)-th argument (or attribute). The set of positions of \( \mathcal{R} \), denoted by \( pos(\mathcal{R}) \), is defined as \( \{ p[i] \mid p/n \in \mathcal{R} \text{ and } i \in [n]\} \). A term \( t \) is a constant, null or variable. An atomic formula (or simply atom) has the form \( p(t) \), where \( p \) is a predicate, and \( t \) a tuple of terms. An atom is called ground if all of its terms are constants of \( C \).

An atom \( a \) refers to its predicate by \( pred(a) \), and we denote by \( dom(a) \), \( var(a) \) and \( pos(a) \) the set of its terms, the set of its variables, and the set of its positions, respectively. Given a set of positions \( \Pi \), we denote by \( var(\Pi) \) the set of variables occurring in \( a \) at positions of \( \Pi \). Furthermore, given a set of variables \( U \), \( pos(\Pi U) \) is the set of positions in \( a \) at which variables of \( U \) occur. The above notations naturally extend to sets of atoms. Conjunctions of atoms are often identified with the sets of their atoms. An instance \( I \) is (a possibly infinite) set of atoms of the form \( p(t) \), where \( t \) is a tuple of constants and nulls. A database \( D \) is a finite instance such that \( dom(D) \subseteq C \).

A substitution from a set of symbols \( S \) to a set of symbols \( S' \) is a function \( h : S \rightarrow S' \) defined as follows: \( \emptyset \) is a substitution (empty substitution), and if \( h \) is a substitution, then \( (h \cup (s \rightarrow s')) \) is a substitution, where \((s, s') \in S \times S' \). The restriction of \( h \) to \( T \subseteq S \), denoted as \( h|_T \), is the substitution \( h|_T = \{ t \mapsto h(t) \mid t \in T \} \). A homomorphism from a set of atoms \( A \) to a set of atoms \( A' \) is a substitution \( h : (C \cup N \cup \mathcal{V}) \rightarrow (C \cup N \cup \mathcal{V}) \) such that: \( t \in C \) implies \( h(t) = t \), and \( r(t_1, \ldots, t_n) \in A \) implies \( h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n)) \in A' \).

A tuple-generating dependency (TGD) \( \sigma \) is a first-order formula \( \forall X \forall Y \exists Z \phi(X, Y, Z) \rightarrow \exists Z \psi(X, Z) \), where \( (X \cup Y \cup Z) \subseteq \mathcal{V} \), and \( \phi, \psi \) are conjunctions of atoms; \( \phi(X, Y) \) is the body of \( \sigma \), denoted \( body(\sigma) \), while \( \psi(X, Z) \) is the head of \( \sigma \), denoted \( head(\sigma) \). The frontier of \( \sigma \), denoted \( fr(\sigma) \), is the set of variables \( X \), and we define \( fpos(\sigma) \) as the set of positions \( pos(head(\sigma), fr(\sigma)) \). Let also \( ex(\sigma) = Z \). Assuming that \( head(\sigma) = a_1 \wedge \ldots \wedge a_k \), let


<table>
<thead>
<tr>
<th>General</th>
<th>Bounded Arity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Linear</td>
<td>NL-c</td>
</tr>
<tr>
<td>Linear</td>
<td>PSPACE-c</td>
</tr>
<tr>
<td>Guarded</td>
<td>2EXPTIME-c</td>
</tr>
<tr>
<td>Weakly-Guarded</td>
<td>2EXPTIME-c</td>
</tr>
</tbody>
</table>

Table 1: Complexity of (Semi-)Oblivious Chase Termination.
(σ, i), where i ∈ [k], be the single-head TGD body(σ) → a_i. The schema of set Σ of TGDs, denoted sch(Σ), is defined as the set of predicates occurring in Σ. An instance I satisfies σ, written I |= σ, if the following holds: whenever there exists a homomorphism h such that h(σ(X, Y)) ⊆ I, then there exists h' ⊇ h_X such that h'ψ(X, Z) ⊆ I. The instance I satisfies a set Σ of TGDs, written I |= Σ, if I |= σ for each σ ∈ Σ. For brevity, we omit the universal quantifiers in front of TGDs, and use the comma (instead of ∧) for conjointing atoms. For technical clarity, we focus our attention on constant-free TGDs.

A TGD σ is guarded if there exists an atom a ∈ body(σ) that contains (or “guards”) all the variables of body(σ) [6]. The class of guarded TGDs, denoted G, is defined as the family of all possible sets of guarded TGDs. Weakly-guarded TGDs extend guarded TGDs by requiring only the body-variables that appear at affected positions, i.e., positions that can host nulls during the chase, to appear in the guard; for the formal inductive definition of affected positions see [6]. The corresponding class is denoted WG. We write guard(σ) for the guard of a (weakly-)guarded TGD σ. A key subclass of G are the so-called linear TGDs [7], that is, TGDs with just one body-atom (which is automatically a guard), and the corresponding class is denoted L. A set of linear TGDs is called simple if there is no repetition of variables in the body of the TGDs, and the corresponding class is denoted SL. It is straightforward to verify that SL ⊆ L ⊆ G ⊆ WG.

2.2 (Semi-)Oblivious TGD Chase Procedure

The TGD chase procedure (or simply chase) takes as input an instance I and a set Σ of TGDs, and constructs a universal model of I and Σ. The chase works on I by applying the so-called trigger for a set of TGDs on I. In what follows, fix a set Σ of TGDs and an instance I

Definition 1. A trigger for Σ on I is a pair (σ, h), where σ ∈ Σ and h is a homomorphism such that h(body(σ)) ⊆ I. An application of (σ, h) to I returns J = (I ∪ h'(head(σ)) where h' ⊇ h_x is such that, for each existentially quantified variable Z ∈ ex(σ), h'(Z) ∈ N does not occur in I, and follows lexicographically all nulls in I. A trigger application is written as I(σ, h)J.

The choice of the type of the next trigger to be applied is crucial since it gives rise to different variations of the chase procedure. In this work, we mainly focus our attention on the oblivious [6] and semi-oblivious [15, 22] chase.

Oblivious. A finite sequence I_0, I_1, . . . , I_n, where n ≥ 0, is said to be a terminating oblivious chase sequence of I_0 w.r.t. a set Σ of TGDs if: (i) for each 0 ≤ i < n, there exists a trigger (σ, h) for Σ on I_i such that I_i(σ, h)I_{i+1}; (ii) for each 0 ≤ i < j < n, assuming that I_i(σ_j, h_j)I_{j+1} and I_j(σ_j, h_j)I_{j+1}, σ_i = σ_j implies h_i ̸= h_j, i.e., h_i and h_j are different homomorphisms; and (iii) there is no trigger (σ, h) for Σ on I_n such that (σ, h) ̸∈ {σ_i, h_i}|0≤i<n. In this case, the result of the chase is the (finite) instance I_n. An infinite sequence I_0, I_1, . . . of instances is said to be a non-terminating oblivious chase sequence of I_0 w.r.t. Σ if: (i) for each i ≥ 0, there exists a trigger (σ, h) for Σ on I_i such that I_i(σ, h)I_{i+1}; (ii) for each i, j ≥ 0 such that i ̸= j, assuming that I_i(σ_j, h_j)I_{j+1} and I_j(σ_j, h_j)I_{j+1}, σ_i = σ_j implies h_i ̸= h_j; and (iii) for each i ≥ 0, and for every trigger (σ, h) for Σ on I_i, there exists j ≥ i such that I_j(σ, h)I_{j+1}; this is known as the fairness condition, and guarantees that all the triggers eventually will be applied. The result of the chase is ∪_i≥0I_i.

Semi-oblivious. The semi-oblivious chase is a refined version of the oblivious chase, which avoids the application of some superfluous triggers. Roughly speaking, given a TGD σ, for the semi-oblivious chase, two homomorphisms h and g that agree on the frontier of σ, i.e., h|fr(σ) = g|fr(σ), are indistinguishable. To formalize this, we first define the binary relation ≈ on the set of homomorphisms H_{σ} = {h | h : var(body(σ)) → (Σ ∪ N)} as follows: h ̸≈ g iff h|fr(σ) = g|fr(σ). It is easy to verify that ≈ is an equivalence relation on the elements of H_{σ}. A (terminating or non-terminating) oblivious chase sequence I_0, I_1, . . . is called semi-oblivious if the following holds: for every i, j ≥ 0 such that i ̸= j, assuming that I_i(σ_i, h_i)I_{i+1} and I_j(σ_j, h_j)I_{j+1}, σ_i = σ_j implies h_i ̸≈ h_j, i.e., h_i and h_j belong to different equivalence classes.

Henceforth, we write CHASE and so-CHASE for oblivious and semi-oblivious chase, respectively.

3. CHASE TERMINATION PROBLEM

We know that, due to the existentially quantified variables, a ∗-chase sequence, where ∗ ∈ {o, so}, may be infinite.

Example 2. Consider the database D = {p(a, b)}, and

Σ = {p(X, Y) → ∃Z p(Y, Z)}.

There exists only one ∗-chase sequence of D w.r.t. Σ, where ∗ ∈ {o, so}, which is non-terminating, i.e., I_0, I_1, . . . with

I_0 = \{p(a, b)\}
I_1 = \{p(a, b), p(b, z_1)\}
I_i = I_{i-1} \{p(z_{i-1}, z_i)\}, for i ≥ 2,

where z_1, z_2, . . . are nulls of N.

For a set of TGDs, a key question is whether all or some ∗-chase sequences are terminating on all databases. Before formalizing the above problem, let us recall the following crucial classes of TGDs:

\[CT_{o} = \{Σ | ∀D, (∃∗-sequences of D w.r.t. Σ are terminating)\}\]
\[CT_{so} = \{Σ | ∀D, there is a terminating ∗-chase sequence of D w.r.t. Σ\}\]

The decision problems tackled in this work are as follows:

\[∀-SEQUENCE ∗-CHASE TERMINATION:\]
Instance: A set Σ of TGDs.
Question: Does Σ ∈ CT_{o}?

\[∃-SEQUENCE ∗-CHASE TERMINATION:\]
Instance: A set Σ of TGDs.
Question: Does Σ ∈ CT_{so}?

It would be quite beneficial for our later investigation to understand how the above problems are related. To this aim, we recall that CT_{o} = CT_{so} ⊆ CT_{[6]} = CT_{3} [15]. This implies that the preceding decision problems coincide for the (semi-)oblivious chase. Henceforth, we refer to the ∗-chase termination problem, and we write CT_{∗} for the classes CT_{o} and CT_{so}, where ∗ ∈ {o, so}.

Another useful notion is the so-called critical database for a set of TGDs [22]. Formally, the critical database for a schema Σ is the database Dc(Σ) = p(c, . . . , c) | p ∈ Σ and c ∈ C}. The critical database for a set Σ of TGDs is defined as the database Dc(sch(Σ)); for brevity, we will refer to Dc(sch(Σ)) by Dc(Σ). To check for the termination of the (semi-)oblivious chase it suffices to focus on the critical database [22].
4. LINEARITY

We proceed to investigate the (semi-)oblivious chase termination problem for (simple) linear TGDs. The goal of this section is twofold: for every $* \in \{0, so\}$,

1. Syntactically characterize the classes $(CT^* \cap SL)$ and $(CT^* \cap L)$; and
2. Pinpoint the complexity of the $*$-chase termination problem for sets of TGDs of $(SL)$.

For our first goal, we are going to exploit existing syntactic conditions that guarantee the termination of every (semi-)oblivious chase sequence on all databases; in fact, our analysis will build on rich-acyclicity [19] and weak-acyclicity [11]. More precisely, we are going to show that for simple linear TGDs rich-acyclicity (resp., weak-acyclicity) is enough for characterizing $(CT^* \cap SL)$ (resp., $(CT^* \cap SL)$). However, for (non-simple) linear TGDs this is not the case, and we need to carefully extend rich- and weak-acyclicity. The above syntactic characterizations, apart from being interesting in their own right, allow us to obtain optimal upper bounds for the $*$-chase termination problem for $(SL)$, and thus achieving our second goal — we simply need to analyze the complexity of deciding whether a set of (simple) linear TGDs enjoys the above acyclicity-based conditions. But let us first recall those conditions.

Weak- and Rich-Acylicity

Both weak- and rich-acyclicity are defined via an acyclicity condition on a graph, which encodes how terms are propagated among the positions of the underlying schema during the chase. In fact, weak-acyclicity uses the well-known dependency graph [11], while rich-acyclicity the so-called extended dependency graph [19]. In the sequel, we assume a fixed order on the head-atoms of TGDs.

Definition 2. The dependency graph of a set $\Sigma$ of TGDs is a labeled directed multigraph $DG(\Sigma) = (N, E, \lambda)$, where $N = pos(sch(\Sigma))$, $N = \{0, so\}$, $\lambda : E \rightarrow \Sigma \times N$, and the edge-set $E$ is as follows:

1. For each $e \in E$, for each $V \in fr(\sigma)$, and for each $\pi \in pos(\text{head}(\sigma), V)$, with $\text{head}(\sigma) = \{0, so\}$, $\ldots \{0\}$.
   - For each $i \in [k]$, and for each $\pi' \in pos(\pi, V)$, there is a normal edge $e = (\pi, \pi') \in E$ with $\lambda(e) = (\sigma, i)$;
   - For each $W \in ex(\sigma)$, for each $i \in [k]$, and for each $\pi' \in pos(\pi, W)$, there is a special edge $e = (\pi, \pi') \in E$ with $\lambda(e) = (\sigma, i)$;
   - No other edges are in $E$.

A normal edge $(\pi, \pi')$ in the dependency graph keeps track of the fact that a term may propagate from $\pi$ to $\pi'$ during the chase. Moreover, a special edge $(\pi, \pi')$ keeps track of the fact that propagation of a value from $\pi$ to $\pi'$ also creates a null at position $\pi'$.

Example 3. Consider the set $\Sigma$ consisting of the TGD

$$\sigma = p(X, Y) \rightarrow \exists Z s(X, Z), p(X, Z).$$

The graph $DG(\Sigma)$ is depicted in Figure 1(a), where the dashed arrows represent special edges. Observe that the normal edges occur due to the variable $X$, while the special edges due to the existentially quantified variable $Z$.

The extended dependency graph of a set $\Sigma$ of TGDs, introduced in [19], is obtained from the dependency graph of $\Sigma$ by adding some additional special edges from the positions where non-frontier variables occur to the positions where existentially quantified variables appear. The extended dependency graph of $\Sigma$ given in Example 2 is shown in Figure 1(b); the additional special edges (dashed arrows) are due to the non-frontier variable $Y$. Having the above structures in place, we can recall weak- and rich-acyclicity. A set $\Sigma$ of TGDs is weakly-acyclic (resp., richly-acyclic) if no cycle in $DG(\Sigma)$ (resp., $EDG(G)$) contains a special edge. The corresponding classes are $WA$ and $RA$, respectively; clearly, $RA \subseteq WA$.

4.1 Characterizing $(CT^0 \cap SL)$ and $(CT^0 \cap SL)$

4.1.1 Oblivious Chase

We start our investigation by showing that rich-acyclicity characterizes the fragment of $SL$ that guarantees the termination of the oblivious chase. In particular, we prove that:

**Theorem 1.** $(CT^0 \cap SL) = (RA \cap SL)$.

To establish the above theorem it suffices to show that, for an arbitrary set of TGDs $\Sigma \in SL$, $\Sigma \in CT^0$ iff $\Sigma \in RA$. The “if” direction has been shown in [19]. Assume now that $\Sigma \not\in RA$. We are going to show that there exists a database $D$, and a non-terminating $*$-chase sequence of $D$ w.r.t. $\Sigma$, which immediately implies that $\Sigma \not\in CT^0$. But let us first introduce our generic technical tool, which will be used also for the semi-oblivious chase, and all the other languages that we treat in this work. Given a TGD $\sigma$, $\sigma_o$ is defined as $\not\in$, if $* = 0$, and $\not\in$, if $* = so$.

**Definition 3.** A set $\Sigma$ of TGDs admits an infinite $*$-chase derivation, where $* \in \{0, so\}$, if there exist infinite sequences $I_0, I_1, \ldots$ and $(\sigma_0, h_0), (\sigma_1, h_1), \ldots$, where $I_0$ is a database, and $\sigma_0, \sigma_1, \ldots \in \Sigma$, such that

1. for each $i \geq 0$, $I_i(\sigma_i, h_i)I_{i+1}$; and
2. for each $i \not\in j \geq 0$, $\sigma_i = \sigma_j$ implies $h_i o_h h_j$.

It is possible to show that the $*$-chase termination problem, where $* \in \{0, so\}$, is tantamount to the problem of deciding whether a set of TGDs admits an infinite $*$-chase derivation.

**Proposition 2.** Consider a set $\Sigma$ of TGDs. $\Sigma \not\in CT^*$ iff $\Sigma$ admits an infinite $*$-chase derivation, where $* \in \{0, so\}$.

The “only-if” direction is trivial. For the “if” direction, it suffices to show that there exists a database $D$, and a non-terminating $*$-chase sequence of $D$ w.r.t. $\Sigma$. By hypothesis, we have sequences $I_0, I_1, \ldots$ and $(\sigma_0, h_0), (\sigma_1, h_1), \ldots$ as in Definition 3. A non-terminating $*$-chase sequence of $I_0$ w.r.t. $\Sigma$ is

$$J_0, J_1^0, \ldots, J_0^k, J_1^1, \ldots, J_1^k, J_2, \ldots$$

where,

- $J_0 = I_0$;
- for each $i \geq 0$, there exists a trigger $(\sigma, h)$ for $\Sigma$ on $J_i$ such that $J_i(\sigma, h)J_{i+1}$;
- for each $i \geq 0$ and $1 \leq j < k_i$, there exists a trigger $(\sigma, h)$ for $\Sigma$ on $J_i$ such that $J_{i}^1(\sigma, h)J_{i+1}^1$.
- for each \( i \geq 0 \), \( J_i^t (\sigma_i, h_i) J_{i+1} \); recall that \( (\sigma_i, h_i) \) is a trigger occurring in the sequence obtained by hypothesis;
- for each pair of triggers \((\sigma, h)\) and \((\sigma', h')\) considered above, \( \sigma = \sigma' \) implies \( h \oplus h' \); and
- for each \( i \geq 0 \), \( k_i \geq 0 \) is the maximal integer such that the above conditions hold.

Roughly, the above chase sequence constructs the chase in a level-by-level fashion, where level zero is defined as \( J_0 \), and the atoms of level \( i \) are obtained by applying TGDs on atoms of level \( i-1 \), by giving priority to the triggers \((\sigma_0, h_0), (\sigma_1, h_1), \ldots \). Thus, the fairness condition is guaranteed, and Proposition 2 follows.

We proceed with the proof of Theorem 1. Recall that we need to show the following: for the set \( \Sigma \in SL \), \( \Sigma \not\in RA \) implies \( \Sigma \not\in CT^o \). By Proposition 2, it suffices to show that, if \( \Sigma \not\in RA \), then \( \Sigma \) admits an infinite o-chase derivation. The rest of this section is devoted to establish that indeed \( \Sigma \) admits an infinite o-chase derivation.

By hypothesis, there exists a cycle in \( EDG(\Sigma) \) that contains a special edge; let \((v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n)\) be such a cycle \((v_0 = v_n)\) with \( \lambda(v_i, v_{i+1}) = (\sigma, k_i)\), for each \( 0 \leq i < n \). In the sequel, we refer to the above cycle by \( C \). One may claim that, starting from a database \( D \) that triggers the TGD \( \sigma_0 \), the cycle \( C \) will give rise to an infinite o-chase derivation, which in turn implies that \( \Sigma \) admits an infinite o-chase derivation, as needed. However, such a derivation may be invalid due to the fact that the involved triggers are not distinct. There is no guarantee that the edges of \( C \) that are labeled with the same TGD give rise to different triggers.

**Example 4.** Consider the set \( \Sigma' \in SL \) consisting of

\[
\begin{align*}
\rho_1 &= p(X, Y) \rightarrow \exists Z s(Z, Z) \\
\rho_2 &= s(X, Y) \rightarrow p(Y, X) \\
\rho_3 &= p(X, Y) \rightarrow s(X, Y).
\end{align*}
\]

It is easy to verify that the cycle depicted in Figure 2(a), where the dashed arrow represents a special edge, occurs in \( EDG(\Sigma') \). Starting from \( I_0 = \{ p(e, c) \} \), where \( e \in C \), if we apply the TGDs as dictated by this cycle, we get an infinite sequence of instances \( I_0, I_1, \ldots \) with

\[
\begin{align*}
I_1 &= I_0 \cup \{ s(z_1, z_1) \} \\
I_2 &= I_3 = I_4 = I_5 \cup \{ p(z_1, z_1) \} \\
I_6 &= I_7 = I_8 = I_9 \cup \{ p(z_2, z_2) \} \\
\ldots
\end{align*}
\]

where \( z_1, z_2, \ldots \) are nulls. However, this sequence is not a valid o-chase derivation since, for each \( i \in \{ 1, 5, 9, 13, \ldots \} \), assuming that \( I_{i+1}(p_2, h) I_{i+3} \) and \( I_{i+2}(p_2, h') I_{i+3}, h = h' = \{ X_2 \rightarrow \tilde{z}_{14}^i, Y_2 \rightarrow \tilde{z}_{14}^i \} \). Thus, \( (p_2, h), (p_2, h') \) are not distinct, as required by an infinite o-chase derivation.

Although \( C \) does not necessarily encode a valid infinite o-chase derivation, it is possible to show that in \( EDG(\Sigma) \) there exists a cycle \( C' \), whose length is less or equal than the length of \( C \), which encodes a valid infinite o-chase derivation. Intuitively speaking, if we avoid to reapply the repeated triggers that are involved in the infinite sequence of instances obtained due to \( C \), then we get a valid o-chase derivation, which corresponds to \( C' \). In fact, \( C' \) is one of the shortest cycles in \( EDG(\Sigma) \) that contains a special edge. Let us illustrate this via an example that builds on Example 4.

**Example 5.** Consider the set \( \Sigma' \) given in Example 4. As already discussed above, starting from \( I_0 = \{ p(e, c) \} \), and applying the TGDs as dictated by the cycle of \( EDG(\Sigma) \) shown in Figure 2(a), we obtain an infinite sequence of instances that is not a valid o-chase derivation, since some of the involved triggers are repeated. If we avoid to reapply those triggers, then we get an infinite sequence of instances \( J_0 = I_0, J_1, \ldots \) with

\[
\begin{align*}
J_1 &= J_0 \cup \{ s(z_1, z_1) \} \\
J_2 &= J_1 \cup \{ p(z_1, z_1) \} \\
J_3 &= J_2 \cup \{ s(z_2, z_2) \} \\
J_4 &= J_3 \cup \{ p(z_2, z_2) \} \\
\ldots
\end{align*}
\]

where \( z_1, z_2, \ldots \) are nulls of \( N \). It is easy to verify that \( J_0, J_1, \ldots \) is a valid infinite o-chase derivation, and that this derivation corresponds to the cycle of \( EDG(\Sigma') \) depicted in Figure 2(b). This cycle is of length two, and there is no shorter cycle that contains a special edge.

From the above discussion, one can exploit the minimal cycles in the extended dependency graph, and show that:

**Lemma 3.** For every set \( \Sigma \in SL \), if \( \Sigma \not\in RA \), then \( \Sigma \) admits an infinite o-chase derivation.

By Proposition 2 and Lemma 3, we immediately get that \( \Sigma \not\in CT^o \), and Theorem 1 follows.

### 4.1.2 Semi-Oblivious Chase

By following a similar approach, we can characterize the fragment of \( SL \) that guarantees the termination of the semi-oblivious chase. For a set of TGDs \( \Sigma \in (RA \cap SL) \) implies \( \Sigma \in (CT^o \cap SL) \), which in turn implies \( \Sigma \in (CT^{op} \cap SL) \). However, the other direction is, in general, not true. Consider the set \( \Sigma \) given in Example 3. It is easy to verify that \( \Sigma \in (CT^{op} \cap SL) \), but \( \Sigma \not\in (RA \cap SL) \), since in its extended dependency graph, which is depicted in Figure 1, there exists a cycle that contains a special edge.

The main reason why rich-acyclicity is not enough for characterizing \( (CT^{op} \cap SL) \), is the existence (in the extended dependency graph) of the special edges from the positions where non-frontier variables occur to the positions where existentially quantified variables appear. In fact, those edges encode erroneous propagations of nulls that do not take place during the construction of the semi-oblivious chase. Recall that after eliminating those problematic special edges, we get a graph structure that coincides with the dependency graph. This observation led us to conjecture that weak-acyclicity is enough for characterizing \( (CT^{op} \cap SL) \). By giving a proof similar to that of Lemma 3, with the difference that we exploit the dependency graph instead of the extended dependency graph, we show that:

**Lemma 4.** For every set \( \Sigma \in SL \), if \( \Sigma \not\in WA \), then \( \Sigma \) admits an infinite o-chase derivation.

By Proposition 2 and Lemma 4, we immediately get that \( \Sigma \not\in WA \) implies \( \Sigma \not\in CT^{op} \). Notice that the other direction is implicit in [22], where the same has been shown for a superclass of WA, and the next result follows:

**Theorem 5.** \( (CT^{op} \cap SL) = (WA \cap SL) \).
Figure 3: Extended dependency graphs of Examples 6 and 7.

4.1.3 Consequences to Other Formalisms

Despite their simplicity, simple linear TGDs are powerful enough for capturing prominent database dependencies, and in particular inclusion dependencies; see, e.g., [1]. It is well-known that inclusion dependencies correspond to simple linear (constant-free) TGDs with just one head-atom without repetition of variables(15,8),(619,896) and we refer to this formalism by ID. Furthermore, simple linear TGDs generalize prominent ontology languages, and in particular DL-Lite[8]. In fact, DL-Lite[8] (ignoring non-membership and disjunctive axioms) corresponds to simple linear TGDs that use only unary and binary predicates; we refer by DL-LiteGD to this formalism. It is evident that our preceding results on simple linear TGDs immediately imply the following:

**Corollary 6.** It holds that,

1. \( (CT^* \cap ID) = (L(\ast) \cap ID) \),
2. \( (CT^* \cap DL-LiteGD) = (L(\ast) \cap DL-LiteGD) \),

where \( \ast \in \{o, so\} \), \( L(o) = RA \), and \( L(so) = WA \).

4.2 Characterizing \((CT^o \cap L)\) and \((CT^o \cap L)\)

4.2.1 Oblivious Chase

We proceed with the characterization of the fragment of L that guarantees the termination of the oblivious chase. Let us first expose, by means of simple examples, the two reasons for which rich-acyclicity is not enough for our purposes.

**Example 6.** Consider the set \( \Sigma \in L \) consisting of

\[
\begin{align*}
\sigma_1 &= p(X, X) \rightarrow \exists Z s(Z, X) \\
\sigma_2 &= s(X, X) \rightarrow \exists Z p(Z, X).
\end{align*}
\]

It is easy to verify that in the extended dependency graph of \( \Sigma \), depicted in Figure 3(a), there exists a cycle that contains a special edge. However, for every database \( D \), every o-chase sequence of \( D \) w.r.t. \( \Sigma \) is terminating.

As shown above, the first reason why rich-acyclicity is not enough for characterizing \((CT^* \cap L)\), is the fact that a cycle in the extended dependency graph does not necessarily encode a chase derivation. Consider, for example, the cycle \((p[1], s[1]), (s[1], p[1])\), where the first edge is labeled by \( \sigma_1 \), and the second edge by \( \sigma_2 \). One expects that, after applying \( \sigma_2 \) during the chase, the obtained atom \( a \) may trigger \( \sigma_2 \). However, this is not the case, since \( a \) is necessarily of the form \( s(t, t') \), where \( t \neq t' \), which means that there is no homomorphism from \( body(\sigma_2) \) to \( a \). The atom \( a \) is of the above form since, in the head-atom of \( \sigma_1 \), at position \( s[1] \) we have an existentially quantified variable, while at position \( s[2] \) a frontier variable.

The above informal discussion, demonstrates the need of finding an effective way for guaranteeing that a cycle in the extended dependency graph can indeed be traversed during the construction of the chase, in which case is called active. To this end, we need to understand when, for two single-head linear TGDs \( \sigma_1 \) and \( \sigma_2 \), the atom obtained by applying \( \sigma_1 \) may trigger \( \sigma_2 \). Notice that we focus on single-head TGDs, since, by definition, the edges of the extended dependency graph are labeled by single-head TGDs. The above property is captured by the notion of compatibility. In the sequel, we assume the reader is familiar with the notion of unification. Given two atoms \( a \) and \( b \) that unify, we denote by MGU\((a, b)\) their most general unifier.

**Definition 4.** Let \( \sigma_1 \) and \( \sigma_2 \) be single-head linear TGDs. Then, \( \sigma_1 \) is compatible with \( \sigma_2 \) if: head\((\sigma_1)\) and body\((\sigma_2)\) unify, and for each \( X \in \text{var(body(\sigma_2))} \), assuming \( \Pi = \text{pos(body(\sigma_2), \{X\})} \), either \( \text{var(head(\sigma_1), \Pi)} \subseteq \text{fr(\sigma_1)} \), or, \( \text{var(head(\sigma_1), \Pi)} = \{Z\} \), for some \( Z \in \text{ex(\sigma_1)} \).

Clearly, in Example 6, \( \sigma_1 \) is not compatible with \( \sigma_2 \), and vice-versa. Having the notion of compatibility in place, one may be tempted to claim that a sequence \( \sigma_1, \ldots, \sigma_n \) of single-head linear TGDs is active if, for each \( i \in [n-1] \), \( \sigma_i \) is compatible with \( \sigma_{i+1} \). However, this does not capture our intention. Instead, we need to ensure that the resolvent of such a sequence, which is a single-head linear TGD that simulates the behavior of \( \sigma_1, \ldots, \sigma_n \), exists.

**Definition 5.** The resolvent of a sequence \( \sigma_1, \ldots, \sigma_n \) of single-head linear TGDs, denoted \( R(\sigma_1, \ldots, \sigma_n) \), is inductively defined as follows (for convenience, we write \( \rho \) for \( \rho(\sigma_1, \ldots, \sigma_{i-1}) \)):

1. \( R(\sigma_1) = \sigma_1 \); and
2. \( R(\sigma_1, \ldots, \sigma_n) = \theta(\text{body}(\rho)) \rightarrow \theta(\text{head}(\sigma_n)) \), where \( \theta = \text{MGU(\text{head}(\rho), \text{body}(\sigma_n))} \) if \( \rho \neq \perp \) and \( \rho \) is compatible with \( \sigma_n \); otherwise, \( R(\sigma_1, \ldots, \sigma_n) = \perp \).

The sequence \( \sigma_1, \ldots, \sigma_n \) is active if \( R(\sigma_1, \ldots, \sigma_n) \neq \perp \).

Apparently, in order to achieve our goal, we need to extend rich-acyclicity to active-rich-acyclicity, by allowing cycles with special edges to appear in the extended dependency graph, as long as they do not give rise to active sequences of single-head linear TGDs. Unfortunately, active-rich-acyclicity is not still not expressive enough for characterizing the fragment of linear TGDs that guarantees the termination of the oblivious chase.

**Example 7.** Consider the set \( \Sigma \in L \) consisting of

\[
\begin{align*}
\sigma_1 &= p(X, Y, Z) \rightarrow s(X, Y, Z) \\
\sigma_2 &= s(X, Y, X) \rightarrow \exists Z p(Y, Z, X).
\end{align*}
\]

It is easy to verify that in EDG\((\Sigma)\), depicted in Figure 3(b), there exists an active cycle that contains a special edge. For example, \( C = (p[2], s[2]), (s[2], p[2]) \) gives rise to the sequence \( \sigma_1, \sigma_2 \). Since \( \sigma_1 \) is compatible with \( \sigma_2 \), we get that \( R(\sigma_1, \sigma_2) \neq \perp \), which in turn implies that \( C \) is active. Even if \( \Sigma \) is not actively-rich-acyclic, we can show that, for every \( D \), every o-chase sequence of \( D \) w.r.t. \( \Sigma \) is terminating. To this aim, it suffices to verify that every o-chase sequence of the critical database \( D_c(\{p, s\}) \) w.r.t. \( \Sigma \) is terminating.

The above example exposes the second reason why rich-acyclicity is not expressive enough for characterizing \((CT^o \cap L)\). In particular, even if a cycle in the extended dependency graph is active, which means that it can be traversed at least once during the construction of the chase, it is not guaranteed that it can be traversed infinitely many times, and thus give rise to an infinite chase derivation. Consider, for example, the cycle \( C = (p[2], s[2]), (s[2], p[2]) \), where the first edge is labeled by \( \sigma_1 \), and the second edge by \( \sigma_2 \). Since \( C \) is active, one expects that, starting from \( p(c, c, c) \), where \( p \) is the predicate of \( body(\sigma_1) \), we can apply \( \sigma_1, \sigma_2, \sigma_1, \ldots \) infinitely many times.
times during the chase. However, after applying \( \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n \), we obtain \( a = s(t, t', e) \), where \( t \neq t' \), and thus there is no homomorphism from body of \( \sigma_2 \) to \( a \). In other words, the cycle \( C \) can be traversed twice, but during its third traversal \( \sigma_3 \) is not triggered. The reason for this behavior is the fact that the sequence \( R(\sigma_1, \sigma_2), R(\sigma_1, \sigma_2), R(\sigma_1, \sigma_2) \) of length three (the chase derivation is blocked during the third traversal of \( C \)) is not active.

It is clear that we need an efficient way for ensuring that an active cycle in the extended dependency graph can be traversed infinitely many times during the construction of the chase. In particular, assuming that an active cycle is labeled by the TGDs \( \sigma_1, \ldots, \sigma_n \), we need to ensure that, for every \( k > 0 \), if \( \rho = R(\sigma_1, \ldots, \sigma_n) \), the sequence \( \rho, \ldots, \rho \) of length \( k \) is active, in which case \( \sigma_1, \ldots, \sigma_n \) is critical. Interestingly, as we shall see below, for ensuring the above criticality condition, we only need to consider sequences of length up to \( (\omega + 1) \), where \( \omega \) is the arity of the predicate of body of \( \sigma_1 \). This leads to the following definition of critical sequences. Henceforth, \( a^k \) denotes the sequence \( \sigma_1, \ldots, \sigma_n \) of length \( k \):

**Definition 6.** A sequence \( \sigma_1, \ldots, \sigma_n \) of single-head linear TGDs is critical if:
1. It is active; and
2. For each \( k \in [\omega + 1] \), \( \rho^k \), where \( \rho = R(\sigma_1, \ldots, \sigma_n) \), is active.

We proceed to extend rich-acyclicity to critical-rich-acyclicity, which, as we shall see, characterizes \((CT^o \cap L)\).

**Definition 7.** Consider \( \Sigma \in L \), and let \( EDG(\Sigma) = (N, E, \lambda) \). A cycle \((v_0, v_1), \ldots, (v_{n-1}, v_n) \) in \( EDG(\Sigma) \) is called critical, if \( \lambda((v_0, v_1)), \ldots, \lambda((v_{n-1}, v_n)) \) is critical. \( \Sigma \) is critically-rich-acyclic: if no critical cycle in \( EDG(\Sigma) \) contains a special edge, and the corresponding class is denoted \( L_{\text{CriticalRA}} \).

The main result of this section follows:

**Theorem 7.** \((CT^o \cap L) = L_{\text{CriticalRA}}\).

The “if” direction of the above result is shown by giving a proof similar to the one given in [19] for showing that \( \Sigma \in RA \) implies \( \Sigma \in CT^o \). The interesting part is to show that, for a set \( \Sigma \in L \), \( \Sigma \not\in L_{\text{CriticalRA}} \) implies \( \Sigma \not\in CT^o \). By Proposition 2, it suffices to show that, if \( \Sigma \not\in L_{\text{CriticalRA}} \), then \( \Sigma \) admits an infinite \( \omega \)-chase derivation. This is a rather non-trivial task, which requires some intermediate results.

The equality type of an atom is a set of equalities among positions that perfectly describe its shape. Formally, given a (constant-free) atom \( a = p(X_1, X_2, \ldots, X_n) \), the equality type of \( a \) is defined as \( \text{eqtype}(a) = \{ p[i] = p[j] \mid X_i = X_j \} \). For a linear TGD \( \sigma \), let \( \text{eqtype}(\sigma) = \text{eqtype}(	ext{body}(\sigma)) \). The following lemma establishes a useful property about active sequences and equality types:

**Lemma 8.** Let \( \sigma \) be a single-head linear TGD such that \( \sigma^i \) and \( \sigma^{i+1} \) are active, for some integer \( i > 0 \), and \( \text{eqtype}(R(\sigma^i)) = \text{eqtype}(R(\sigma^{i+1})) \). Then, \( \sigma^{i+2} \) is active, and \( \text{eqtype}(R(\sigma^{i+1})) = \text{eqtype}(R(\sigma^{i+2})) \).

The above result allows us to show that critical cycles can be traversed infinitely many times during the construction of the chase, starting from the critical database. Consider a critical sequence \( \sigma_1, \ldots, \sigma_n \) of single-head linear TGDs, and let \( \rho = R(\sigma_1, \ldots, \sigma_n) \).

It is not difficult to show that there exists \( i \in [\omega + 1] \) such that \( \rho^i \) and \( \rho^{i+1} \) are active, and \( \text{eqtype}(R(\rho^i)) = \text{eqtype}(R(\rho^{i+1})) \). By recursively applying Lemma 8, we conclude that, for every \( k > [\omega + 1] \), \( \rho^k \) is active. Moreover, since \( \sigma_1, \ldots, \sigma_n \) is critical, for every \( k \in [\omega + 1] \), \( \rho^k \) is active. From the above discussion, we get the following crucial result:

**Lemma 9.** Let \( \sigma_1, \ldots, \sigma_n \) be a critical sequence of single-head linear TGDs. Then, for every \( k > 0 \), \( \rho^k \), where \( \rho = R(\sigma_1, \ldots, \sigma_n) \), is active.

By using Lemma 9, and the fact that the resolvent of an active sequence of single-head linear TGDs mimics the behavior of the sequence during the chase, starting from the critical database (this can be easily shown by induction on the length of the sequence), we can establish that a minimal critical cycle that contains a special edge gives rise to an infinite chase derivation, which in turn implies the following:

**Lemma 10.** For every set \( \Sigma \in L \), if \( \Sigma \not\in L_{\text{CriticalRA}} \), then \( \Sigma \) admits an infinite \( \omega \)-chase derivation.

By Proposition 2 and Lemma 10, we get that \( \Sigma \not\in L_{\text{CriticalRA}} \) implies \( \Sigma \not\in CT^o \), and Theorem 7 follows.

### 4.2.2 Semi-Oblivious Chase

By applying similar techniques, we can characterize the fragment of \( L \) that guarantees the termination of the semi-oblivious chase. To this end, we first need to introduce the notion of critical-weak-acyclicity, which is defined as critical-rich-acyclicity, with the difference that the desired condition is posed on the dependency graph, and not on the extended dependency graph.

**Definition 8.** A set \( \Sigma \in L \) is critically-weakly-acyclic, if no critical cycle in \( DG(\Sigma) \) contains a special edge, and the corresponding class is denoted \( L_{\text{CriticalWA}} \).

As already discussed in Section 4.1.2, the extended dependency graph encodes propagations of nulls that do not take place during the construction of the semi-oblivious chase, and this is exactly the reason why we need to rely on the dependency graph for the characterization of \((CT^o \cap L)\). By giving a proof similar to that of Lemma 10, with the difference that we exploit the dependency graph instead of the extended dependency graph, we show that:

**Lemma 11.** For every set \( \Sigma \in L \), if \( \Sigma \not\in L_{\text{CriticalWA}} \), then \( \Sigma \) admits an infinite \( \omega \)-chase derivation.

By Proposition 2 and Lemma 11, \( \Sigma \not\in L_{\text{CriticalWA}} \) implies \( \Sigma \not\in CT^o \). The proof of the other direction is along the lines of the proof given in [11] for showing that weak-acyclicity guarantees the termination of the restricted chase, and we get that:

**Theorem 12.** \((CT^o \cap L) = L_{\text{CriticalWA}}\).

### 4.3 Complexity of Chase Termination

Let us now proceed with our second goal, that is, to pinpoint the complexity of the \( \omega \)-chase termination problem for sets of TGDs of \( \mathcal{S}L \), where \( * \in \{o, so\} \).

#### 4.3.1 Simple Linear TGDs

We first focus on \( SL \), and we show the following:

**Theorem 13.** Consider a set \( \Sigma \in SL \). The problem of deciding whether \( \Sigma \in CT^* \), where \( * \in \{o, so\} \), is \( \text{NSPACE}(\log(\omega \cdot |sch(\Sigma)|)) \), for \( \omega \) is the maximum arity of \( sch(\Sigma) \).

**Upper Bound.** To obtain the upper bound, by Theorems 1 and 5, it suffices to show that deciding whether \( \Sigma \) is richly-acyclic (or weak-acyclic) is in \( \text{NL} \).

**Lemma 14.** Consider \( \Sigma \in SL \). The problem of deciding if \( \Sigma \in \mathcal{L} \), where \( \mathcal{L} \subseteq \{RA, WA\} \), is in \( \text{NSPACE}(\log(\omega \cdot |sch(\Sigma)|)) \), where \( \omega \) is the maximum arity of \( sch(\Sigma) \).
The complement of the problem under consideration can be seen as an instance of graph reachability. In fact, we need to decide whether there exists a node $v$ in the (extended) dependency graph of $\Sigma$ that is reachable from itself, with the additional condition that the path from $v$ to itself contains at least one special edge. This can be done via a nondeterministic procedure, where at each step needs to remember two consecutive edges of the graph (i.e., three positions of $\text{sch}(\Sigma)$, the origin of the traversed cycle (i.e., the position $v$), and a binary value indicating whether a special edge has been visited or not. All the above elements can be maintained in $O(\log(\omega \cdot |\text{sch}(\Sigma)|))$ space.

**Lower Bound.** Let us now proceed with the $\text{NL}$-hardness. We first introduce the so-called looping operator, which will allow us to establish a generic complexity tool for proving lower bounds for the chase termination problem. Notice that this tool will be used, not only for simple linear TGDs, but also for all the other languages considered in this work. In fact, the goal of the looping operator is to provide a generic reduction from propositional atom entailment to the complement of chase termination. Recall that an instance of propositional atom entailment consists of a database $D$, a set $\Sigma$ of TGDs, and a propositional (i.e., 0-ary) predicate $q$, and the question is whether $D \cup \Sigma \models q$, or, equivalently, whether $q$ belongs to the result of the chase of $D$ w.r.t. $\Sigma$.

Let $(D, \Sigma, q)$ be an instance of propositional atom entailment. Given an atom $a = p(t)$, where $t = (t_1, \ldots, t_n)$, occurring either in $D$ (i.e., $t \in V^n$) or in $\Sigma$ (i.e., $t \in V^n$), we define, for some $Y \subseteq V$ not in $\Sigma$, the atomic formula
\[
a^Y_{\Sigma} = \left\{ \begin{array}{ll}
X_{t_1} \cdots X_{t_n} p(Y, X_{t_1}, \ldots, X_{t_n}), & t \in C^n, \\
p(Y, t_1, \ldots, t_n), & t \in V^n,
\end{array} \right.
\]
where $X_{t_1}, \ldots, X_{t_n} \in V$ do not appear in $\Sigma$. We define $\Phi^Y_{D, \Sigma} = (\bigwedge_{a \in D} a^Y_{\Sigma})$, and $\Sigma^Y$ as the set of TGDs obtained by replacing each atom $q$ occurring in $\Sigma$ with $a^Y_{\Sigma}$. We are now ready to define the looping operator.

**Definition 9.** Let $(D, \Sigma, q)$ be an instance of propositional atom entailment. The application of the looping operator on $(D, \Sigma, q)$ returns the set of TGDs
\[
\text{Loop}(D, \Sigma, q) = \{\text{loop}(X, Y) \rightarrow \Phi^Y_{D, \Sigma} \cup \Sigma^Y \cup \{q(Y) \rightarrow \exists Z \text{ loop}(Y, Z)\},
\]
where $\text{loop} \notin \text{sch}(\Sigma)$. A class of TGDs $\mathcal{L}$ is closed under looping if, for every instance $(D, \Sigma, q)$ of propositional atom entailment, where $\Sigma \in \mathcal{L}$, $\text{Loop}(D, \Sigma, q) \in \mathcal{L}$.

By using the looping operator, we can transfer, in a uniform way, lower bounds from propositional atom entailment to chase termination. Our generic complexity result follows:

**Proposition 15.** Let $\mathcal{L}$ be a class of TGDs that is closed under looping, such that propositional atom entailment for $(\text{CT}^* \cap \mathcal{L})$, where $\ast \in \{0, \text{so}\}$, is $\text{C}$-hard, for a complexity class $\mathcal{C}$ that is closed under log-space reductions. For a set $\Sigma \in \mathcal{L}$, deciding whether $\Sigma \in \text{CT}^*$ is $\text{coC}$-hard.

To establish the above generic result, it suffices to reduce propositional atom entailment under $(\text{CT}^* \cap \mathcal{L})$ to the complement of chase termination under $\mathcal{L}$. Given a (non-empty) database $D$, a set $\Sigma \in (\text{CT}^* \cap \mathcal{L})$, and a propositional predicate $q$, we need to construct in log-space a set $\Sigma' \in \mathcal{L}$ such that, $D \cup \Sigma \models q$ if and only if there exists a database $D'$ such that a non-terminating $*$-chase sequence of $D'$ w.r.t. $\Sigma'$ exists. It can be shown that the above equivalence holds for $\Sigma' = \text{Loop}(D, \Sigma, q)$. The key idea underlying the looping operator can be sketched described as follows. Consider the simple linear TGD $\sigma = \text{loop}(X, Y) \rightarrow \exists Z \text{ loop}(Y, Z)$. It is easy to verify that there exists only one $*$-chase sequence of $\{\text{loop}(a, b)\}$ w.r.t. $\{\sigma\}$, which is non-terminating; for details, see Example 2. Our intention is to mimic the behavior of $\sigma$ using $\Sigma'$, with the key difference that an atom of the form $\text{loop}(t', t'')$ is obtained by applying $\sigma$ on an atom $\text{loop}(t, t')$ only if $D \cup \Sigma \models q$. This is achieved by “plugging” between $\text{body}(\sigma)$ and $\text{head}(\sigma)$ the set $\Sigma'$, which, by hypothesis, guarantees the termination of the chase. The given database $D$ is generated by the TGD $\text{loop}(X, Y) \rightarrow \Phi^Y_{D, \Sigma}$, while the check whether $q$ is entailed is performed by $q(Y) \rightarrow \exists Z \text{ loop}(Y, Z)$. Since, by assumption, $\mathcal{L}$ is closed under looping, $\Sigma' \in \mathcal{L}$, and Proposition 15 follows.

By the Immerman-Szelepcsenyi theorem, $\text{coNL} = \text{NL}$. Thus, to obtain the $\text{NL}$-hardness for the chase termination problem under simple linear TGDs, since $\mathcal{SL}$ is closed under looping, by Proposition 15, it suffices to show that propositional atom entailment under $(\text{CT}^* \cap \mathcal{SL})$ is $\text{NL}$-hard, even for unary and binary predicates. This is shown by giving a reduction from graph reachability. Given a directed graph $G = (N, E)$, we construct a database $D$, a set $\Sigma \in \mathcal{SL}$, and a propositional predicate $q$ such that $D \cup \Sigma \models q$ if and only if $G$ is reachable from $s$. The idea is to construct $\Sigma$ in such a way that its predicate graph coincides with $G$, while $D$ stores the node $s$, and $q$ represents the node $t$. We get that:

**Lemma 16.** Propositional atom entailment under $(\text{CT}^* \cap \mathcal{SL})$, where $\ast \in \{0, \text{so}\}$, is $\text{NL}$-hard, even for unary and binary predicates.

Theorem 13 follows from Proposition 15, and Lemmas 14 and 16. Notably, $\text{Loop}(D, \Sigma, q)$ belongs to $\text{ID}$ and $\text{DL-Lite}^{\text{GD}}$, and thus Theorem 13 holds also for inclusion dependencies and $\text{DL-Lite}_{\mathcal{R}}$.

### 4.3.2 Linear TGDs

We now focus on arbitrary linear TGDs, and we show the following:

**Theorem 17.** Consider a set $\Sigma \in \mathcal{L}$. The problem of deciding whether $\Sigma \in \text{CT}^*$, where $\ast \in \{0, \text{so}\}$, is $\text{PSPACE}$-complete, and $\text{NL}$-complete for predicates of bounded arity.

**Upper Bound.** By Theorems 7 and 12, it suffices to show that the problem of deciding whether $\Sigma$ is critically-richly-acyclic (or critically-weakly-acyclic) can be solved in polynomial space, in general, and in nondeterministic logarithmic space, in case of predicates of bounded arity.

**Lemma 18.** Consider a set $\Sigma \in \mathcal{L}$. The problem of deciding if $\Sigma \in \mathcal{L}$, where $\mathcal{L} \in \{\text{LCriticalRA, LCriticalWA}\}$, is in $\text{NSPACE}(|\text{log}(\omega \cdot |\text{sch}(\Sigma)|) + \omega \log(\omega \cdot |\Sigma|))$, where $\omega$ is the maximum arity over all predicates of $\text{sch}(\Sigma)$.

The above technical lemma is shown by conceiving the complement of our problem as an extended version of graph reachability. In particular, we need to decide whether there exists a node $v$ in the (extended) dependency graph of $\Sigma$ that is reachable from itself via a critical cycle that contains a special edge. As for Lemma 14, this can be done via a nondeterministic procedure. However, in order to check for the criticality of the traversed cycle, apart from the two consecutive edges, the origin of the cycle, and the binary flag, we also need to remember the resolvent of the TGDs that label the visited edges. Such a resolvent can be computed and maintained in
Theorem 13. Concerning the classes problem for guarded and weakly-guarded TGDs. Although there is standard databases. We show the following:

The results presented below, unless stated otherwise, hold only for databases that have at least two constants, let’s say \( \sigma_0 \) and \( \sigma_1 \), with \( \sigma_0 \) and \( \sigma_1 \) being isomorphic to \( \sigma_0 \) and \( \sigma_1 \), respectively. The results presented below, unless stated otherwise, hold only for standard databases. We show the following:

Theorem 17 follows from Proposition 15, and Lemmas 18 and 19.

5. (WEAK-)GUARDEDNESS

We proceed to investigate the (semi-)oblivious chase termination problem for guarded and weakly-guarded TGDs. Although there is no way (at least no obvious one) to syntactically characterize the guarded TGDs (not necessarily weakly-guarded), denoted \( D_{\text{std}}(\Sigma) \), is defined as the database

\[
\{0(0), 1(1)\} \cup \{p(t) \mid p/n \in \text{sch}(\Sigma), t \in \{0, 1\}^*\}
\]

It is clear that the size of \( D_{\text{std}}(\Sigma) \) is exponential in general, and polynomial when the maximum arity over all predicates of \( \text{sch}(\Sigma) \) is fixed. By giving a proof similar to the one in [22] for the critical database, we show the following:

Lemma 21. Consider a set \( \Sigma \) of TGDs. It holds that, \( \Sigma \) admits an infinite \( * \)-chase derivation that starts from a standard database, where \( * \in \{\sigma, so\} \), if \( \Sigma \) admits an infinite \( * \)-chase derivation that starts from \( D_{\text{std}}(\Sigma) \).

Our alternating algorithm, starting from an atom of \( D_{\text{std}}(\Sigma) \), and applying nondeterministically chase steps, identifies a finite basic block of a chase derivation (if it exists), which can then be repeated and give rise to an infinite chase derivation; this is graphically illustrated in Figure 4(a). In other words, the algorithm tries to identify an atom \( \sigma_0 \), from which, after applying some valid (depending on the version of the chase) triggers, an atom \( \sigma_1 \) isomorphic to \( \sigma_0 \) is obtained — by isomorphism we mean that, starting from \( \sigma_0 \) and \( \sigma_1 \), we obtain isomorphic atoms. The segment of the derivation between \( \sigma_0 \) and \( \sigma_1 \) is the basic block that we can repeat infinitely many times, and obtain an infinite chase derivation. Before giving the technical details of our algorithm, we need to briefly recall some auxiliary notions and results.

Auxiliary Notions and Results. A set \( \Sigma \subseteq \text{WG} \) can be effectively transformed into a set \( \Sigma' \subseteq \text{WG} \) such that all the TGDs of \( \Sigma' \) are single-head [6]. It is not difficult to verify that this transformation preserves chase termination, i.e., \( \Sigma \in \text{CT} \) iff \( \Sigma' \in \text{CT} \), where \( \star \in \{\sigma, so\} \). Henceforth, for technical clarity, we focus on TGDs with just one atom in the head. Let \( D \) be a database, and \( \Sigma \) be a set of TGDs. Fix a \( * \)-chase sequence \( I_0 = D, I_1, \ldots \) of \( D \) w.r.t. \( \Sigma \), for \( * \in \{\sigma, so\} \).

The instance \( \cup_{I_0 \neq I_1} \text{sch}(D, I_0) \), denoted \( \text{sch}(D, \Sigma) \), can be naturally represented as a labeled directed graph \( G = (N, E, \lambda) \) as follows: (1) for each atom \( a \in \text{sch}(D, \Sigma) \), there exists \( v \in N \) such that \( \lambda(v) = a(\cdot) \) for each \( i \geq 0 \), with \( I_i(\sigma, h)I_{i+1} \), and for each atom \( \sigma \in h(\text{body}(h)) \), there exists \( (v, u) \in E \) such that \( \lambda(v) = a \) and \( \{\lambda(u)\} = I_{i+1} \setminus I_i \); and (3) there are no other nodes and edges in \( G \). The guarded chase forest of \( D \) and \( \Sigma \), denoted \( \text{gcd}(D, \Sigma) \), is the forest obtained from \( G \) by keeping only the nodes associated with guard atoms, and their children; for more details, we refer the reader to [6].

Lemma 21 implies that our algorithm has to identify an infinite path in \( \text{gcd}(D_{\text{std}}(\Sigma), \Sigma) \). This is achieved by constructing nondeterministically such a path, starting from an atom of \( D_{\text{std}}(\Sigma) \), until a basic block that can be repeated is identified. During this process, our algorithm exploits two key results established in [6], where the problem of query answering under (weakly-)guarded TGDs is investigated. Let us recall those results, and explain how they are applied; let \( D \) be an arbitrary database:

1. The subtree of \( \text{gcd}(D, \Sigma) \) rooted at some atom \( a \) is determined by the so-called cloud of \( a \) (modulo renaming of nulls) [6, Theorem 5.16]. The cloud of \( a \) w.r.t. \( D \) and \( \Sigma \), denoted

\[
\text{cloud}_a(D, \Sigma)
\]
cloud(a, D, σ), is defined as
\[ \{ b \mid b \in \star\text{-chase}(D, \Sigma) \text{ and } \text{dom}(b) \subseteq (\text{dom}(D) \cup \text{dom}(a)) \} , \]
i.e., the atoms occurring in the result of the chase with constants from \( D \) and terms from \( a \). This result allows us to build the relevant path of gcf(D, Σ). In fact, an atom \( a \)
on this path can be generated by considering only its parent atom \( a' \) and the cloud of \( a' \) w.r.t. \( D \) and \( σ \). Whenever a new atom is generated, we nondeterministically guess its cloud, and verify in a parallel universal computation of our algorithm that indeed belongs to the result of the chase.

2. There exists a bound \( δ \), which is double-exponential in the maximum arity \( ω \) of sch(Σ) (and only \( ω \) appears in the second exponent), up to which we have to construct the relevant path of gcf(D, Σ) in order to guarantee that all the obtained atoms are non-isomorphic. This implies that, for our purposes, we simply need to construct the path up to depth \( (2 \cdot δ) \).

We use this fact to ensure that our algorithm terminates.

Let us clarify that in [6] only the oblivious chase has been considered, and the above results have been explicitly established for the oblivious chase. Nevertheless, it is not difficult to extend these results to the semi-oblivious chase.

**The Alternating Algorithm.** We have now all the ingredients in place that are needed to define our algorithm. Given a set \( Σ = \{ σ_1, \ldots, σ_n \} \) as input, \( \star\text{-InfiniteDerivation}(Σ) \) consists of the following steps:

1. \( \text{CL} := D_{\text{add}}(Σ), \text{H}_σ := \emptyset, \) for each \( i \in [n], \text{flag} := 0 \) and \( \text{ctr} := 0 \).
2. Guess an atom \( a \in D \).
3. Guess a TGD \( σ \in Σ \), and a trigger \((σ, h)\) for \( Σ \) on \( CL \), where \( h(\text{guard}(σ)) = a \) and \( h \circ_σ h' \), for each \( h' \in \text{H}_σ \); if there is no such a trigger, then reject.
4. Let \( a \) be the atom obtained by applying \((σ, h)\) to \( CL \), and guess the cloud \( \text{CL} \) of \( a \) w.r.t. \( D \) and \( Σ \).
5. Universally goto steps 6 and 7.
6. If \( \text{CL} \) is a valid cloud, then accept; otherwise, reject.
7. \( \text{H}_σ := (\text{H}_σ \cup \{ h \}) \setminus \text{H}_{σ,a} \text{ where } \text{H}_{σ,a} \subseteq \text{H}_σ \text{ is the set of homomorphisms that map a variable of } \text{var(body}(σ)) \text{ to a term not in } \text{dom}(a). \)
8. If \( \text{flag} = 0 \), then guess to apply or skip the following:
   (a) \( \text{loop} := (σ, a, CL) \) and \( \text{flag} := 1 \).
   (b) \( \text{nulls} := \text{invent}(a), \) where the latter is the set of nulls invented in \( a \); if \( \text{nulls} = \emptyset \), reject.
   (c) Goto step 10.
9. If \( \text{flag} = 1 \), then do the following:
   (a) \( \text{nulls} := (\text{dom}(a) \cap \text{nulls}) \cup \text{invent}(a). \)
   (b) If \( (\text{dom}(a) \cap \text{nulls}) = \emptyset \), then reject.
   (c) If \( (σ, a, CL) \) and \( \text{loop} \) are the same (modulo bijective null renaming), then accept.
10. If \( \text{ctr} = (2 \cdot δ) \), then reject; otherwise, \( \text{ctr} := \text{ctr} + 1 \) and goto step 3.

By construction, \( \star\text{-InfiniteDerivation}(Σ) \), starting from an atom \( a \in D_{\text{add}}(Σ) \), identifies a basic block on a path \( P \) in the subtree of gcf(D_{\text{add}}(Σ), Σ) rooted at \( a \), which can be repeated infinitely many times. This allows us to safely conclude that \( P \) is an infinite path (or chase derivation), and the algorithm accepts; if such a derivation does not exist, the algorithm terminates and rejects. It remains to explain why this derivation is a valid one, i.e., it does not contain conflicting triggers (depending on the version of the chase).

Consider two triggers \((σ, h) \) and \((σ', h') \) occurring in the obtained infinite \( \star\text{-chase} \) derivation. There are two possible cases: either they occur in the same or in different basic blocks; this is illustrated in Figure 4(b). In the first case, \( h \circ_σ h' \) is guaranteed by construction; this is the reason why the set \( H_0 \) is maintained during the execution of the algorithm, which stores all the “dangerous” homomorphisms that have been used to trigger \( σ \). In the second case, \( h \circ_σ h' \) is guaranteed since the atom \( h(\text{guard}(σ)) \) necessarily contains a null that does not appear in the atom \( h(\text{guard}(σ)) \); this is why the set \( \text{nulls} \) is maintained, which actually stores the nulls that can only appear in the atoms of a certain basic block. From the above discussion, we get the desired result:

**Proposition 22.** Consider a set \( Σ ∈ \mathcal{WG} \). It holds that, \( Σ \) admits an infinite \( \star\text{-chase} \) derivation that starts from a standard database, where \( * ∈ \{0, so\} \), iff \( \star\text{-InfiniteDerivation}(Σ) \) accepts.

### 5.2 Complexity of Chase Termination

#### 5.2.1 Upper Bounds

By Propositions 2 and 22, we get that, for a set \( Σ ∈ \mathcal{WG} \), \( Σ \not\subset CT^* \) iff \( \star\text{-InfiniteDerivation}(Σ) \) accepts, where \( * ∈ \{0, so\} \). Therefore, to establish the desired upper bounds, it suffices to show that our alternating algorithm runs in exponential space, in general, and in polynomial space, in the case of predicates of bounded arity; recall that \( \text{AEXPSPACE} = \text{2EXPTIME} \) and \( \text{APSPACE} = \text{EXPTIME} \).

To this end, we show that the space required for the following tasks is exponential in the maximum arity \( ω \) of \( \text{sch}(Σ) \), and polynomial in all the other parameters of the input: (1) maintain \( D_{\text{add}}(Σ) \) and the cloud of an atom; (2) maintain the set \( H_σ \), where \( σ ∈ Σ \); (3) maintain the integer value of \( \text{ctr} \); and (4) verify that the guessed cloud is valid.

**Lemma 23.** The algorithm \( \star\text{-InfiniteDerivation} \), where \( * ∈ \{0, so\} \), runs in double-exponential time, in general, and in exponential time, for predicates of bounded arity.

The upper bounds of Theorem 20 follow from Propositions 2 and 22, and Lemma 23.

#### 5.2.2 Lower Bounds

To establish the desired lower bounds, since \( G \) is closed under looping, by Proposition 15, it suffices to show the following:

**Lemma 24.** Propositional atom entailment under \((CT^* \cap G)\), where \( * ∈ \{0, so\} \), is \( \text{2EXPTIME} \)-hard, and \( \text{EXPTIME} \)-hard for predicates of bounded arity.

The \( \text{2EXPTIME} \)-hardness is obtained by a significant modification of the proof of Theorem 6.2 in [6], which shows the \( \text{2EXPTIME} \)-hardness of propositional atom entailment under arbitrary guarded TGDs (not necessarily in \( CT^* \)). That proof simulates an \( \text{AEXPSPACE} \) Turing machine that uses no more than \( 2^n \) worktape cells; this assumption can be made without affecting the generality of the proof. For proving Lemma 24, we make, w.l.o.g., an additional assumption: we assume the machine contains a counter of \( 2^{n-1} \) bits (i.e., the second half of the tape) that is initialized to zero and can count from 0 up to \((2^{2n-1} - 1) \). The counter is incremented by one until either the Turing machine stops, or it reaches the maximal value of \((2^{2n-1} - 1) \), in which case the machine is forced to stop in a rejecting state. This makes sure that the machine cannot cycle and always stops within \( O(2^n) \) steps. Adding counters
to Turing machines, giving rise to the concept of \textit{clocked Turing machines}, is a well-known technique; see [20, 25]. The fact that we consider a clocked Turing machine, together with the fact that we focus on standard databases, allows us to construct the double-exponentially many configurations of the machine using a set of TGDs that ensures the termination of the chase, which is not the case in the proof of Theorem 6.2 of [6]. By following a similar approach, we can also show the \textit{EXPTIME}-hardness in the case of predicates of bounded arity, and Lemma 24 follows.

5.3 Non-Standard Databases

From the above discussion, it is clear that standard databases are crucial for establishing the lower bounds in Proposition 24; in particular, to guarantee that the sets of guarded TGDs employed in the reductions are indeed members of \((CT^* \cap G)\), where \(* \in \{o, so\}\). Interestingly, the upper bounds stated in Theorem 20 hold also for non-standard databases. This can be shown by slightly modifying \(*\)-\textit{InfiniteDerivation} in such a way that, instead of starting from \(D_{\text{dat}}(\Sigma)\), where \(\Sigma \in WG\) is the given set of TGDs, starts from the critical database \(D_c(\Sigma)\). After applying this modification, it is easy to see that \(\Sigma\) admits an \(*\)-\textit{chase} derivation (that starts from an arbitrary, not necessarily standard database) iff \(*\)-\textit{InfiniteDerivation}(\(\Sigma\)) accepts, and we immediately get the following result for arbitrary databases:

\textbf{Theorem 25.} Consider a set \(\Sigma \in WG\). The problem of deciding whether \(\Sigma \in CT^*\), where \(* \in \{o, so\}\), is in \textit{EXPTIME}, and in \textit{EXPTIME} for predicates of bounded arity.

The exact complexity of the chase termination problem in case of arbitrary (not necessarily standard databases) is still open.

6. RELATIVE EXPRESSIVE POWER

The results presented above, apart from giving an effective way for deciding the termination of the (semi-)oblivious chase, provide us with new decidable query languages, which are based on guardedness, that can directly exploit the chase procedure. A natural question at this point is whether these new query languages, and in particular the feature of existential quantification, give us more expressive power than existing ones. The goal of the current section is to give an answer to the above crucial question.

Consider a set \(L\) of TGDs. An \(L\) query is a pair \((\Sigma, q)\), where \(\Sigma \in L\), and \(q\) is a predicate that does not occur in the body of a TGD in \(\Sigma\). Given a database \(D\), the answer to \(q = (\Sigma, q)\) over \(D\), with \(q\) be an \(n\)-ary predicate, is defined as the set \(\text{ans}(Q, D) = \{ t \in C^n : \exists \Sigma = q(t) \}\). Given two classes \(L\) and \(L'\) of TGDs, we say that \(L'\) is more expressive than \(L\), and we write \(L \triangleright L'\), if, for every \(L\) query \(Q\), we can construct an \(L'\) query \(Q'\) such that, for every database \(D\), \(\text{ans}(Q, D) = \text{ans}(Q', D)\). Finally, \(L\) and \(L'\) have the same expressive power, written \(L \approx L'\), if \(L \triangleright L'\) and \(L' \triangleright L\). Recall that our goal is to understand whether the existential quantification gives us more expressive power. Towards this aim, we are going to compare the expressive power of the query languages obtained from our previous analysis on chase termination, with the relevant fragments of Datalog. We write \(\text{DAT}\) for the family of all Datalog programs, which can be seen as sets of single-head TGDs without existentially quantified variables, and \(\text{UCQ}\) (unions of conjunctive queries) for the family of all Datalog programs where all the rules have the same head-predicate that does not appear in a rule-body. The main result of this section states that the existential quantification does not add expressive power to the existing Datalog-based languages.

\textbf{Theorem 26.} For each \(* \in \{o, so\}\),

1. \((CT^* \cap L) \approx (\text{UCQ} \cap L)\), where \(L \in \{SL, L\}\); and
2. \((CT^* \cap L) \approx (\text{DAT} \cap L)\), where \(L \in \{G, WG\}\).

The "\(\approx\)" direction is trivial since \(\text{DAT}\) guarantees the termination of the chase. For the "\(\triangleright\)" direction, we exploit recent rewriting techniques proposed in different contexts [13, 14, 23].

\textbf{Succinctness.} The next question that comes up concerns the succinctness of our new query languages. This challenging problem goes beyond the scope of the current work, and is something that we are currently investigating. Nevertheless, we would like to present a result, which can be seen as a strong indication that the existential quantification allows us to build more succinct queries.

\textbf{Proposition 27.} Given a \((CT^* \cap G)\) query \(Q\), where \(* \in \{o, so\}\), there is no \((\text{DAT} \cap G)\) (or even \(\text{DAT}\)) query \(Q'\) that is constructible in polynomial time such that, for every standard database \(D\), \(\text{ans}(Q, D) = \text{ans}(Q', D)\).

Since \textit{EXPTIME} \(\subseteq 2\textit{EXPTIME}\), the above result follows from Lemma 24, and the fact that Datalog is in \textit{EXPTIME} in combined complexity. In simple words, Proposition 27 says that, even if \((CT^* \cap G)\) is not more succinct than \((\text{DAT} \cap G)\), to construct an equivalent \((\text{DAT} \cap G)\) query of polynomial size will be a hard task.

7. CONCLUSIONS

We show that the (semi-)oblivious chase termination problem for guarded-based TGDs is decidable, and we obtain precise complexity bounds. To the best of our knowledge, this is the first work that establishes positive results about the (semi-)oblivious chase termination problem. The next step is to perform similar analysis focusing on the restricted version of the chase. The nondeterministic nature of the restricted chase makes the chase termination problem even more challenging, and new techniques must be devised. We have some preliminary positive results, and we are currently working towards the full settlement of the problem.

8. ACKNOWLEDGMENTS

Part of the work of M. Calautti was performed while visiting the Department of Computer Science, University of Oxford, UK. Part of the work of A. Pieris was performed while was post-doctoral researcher at the Department of Computer Science, University of Oxford, UK. M. Calautti was supported by the European Commission, European Social Fund and Region Calabria. G. Gottlob was supported by the EPSRC Programme Grant EP/M025268/1 "VADA: Value Added Data Systems – Principles and Architecture". A. Pieris was supported by the EPSRC Grant EP/J008346/1, Austrian Science Fund (FWF), projects P25207-N23 and Y698, and Vienna Science and Technology Fund (WWTF), project ICT12-015. We thank the anonymous referees for many helpful comments.

9. REFERENCES


