Resolution And Datalog Rewriting
Under Value Invention And Equality Constraints

Bruno Marnette
marnette.bruno@gmail.com

Disclaimer: This preliminary report was originally written at the INRIA Saclay in 2010. The list of references may therefore be incomplete at the time of this release (Dec 2012)

ABSTRACT

While Datalog is a golden standard for denotational query answering, it does not support value invention or equality constraints. The Datalog\(^+\) framework introduced by Gottlob faces these issues by considering rules with fresh variables in the head (known as tgds) or equalities in the head (known as egds). Several tractable classes have been identified, among which: (S) the class of sticky tgds; (T) the class of tgds and egds ensuring oblivious termination; and (G) the class of guarded tgds. In turn, the tractability of these classes typically relies on the ‘chase’: (S) ensures that every chase derivation is ‘sticky’; (T) ensures polynomial chase termination; and (G) allows stopping the chase after a fixed ‘depth’ while preserving completeness.

This paper shows that there are alternative algorithms (instead of the chase) that can serve as a basis for the design of (larger) tractable classes. As a first contribution, we present an algorithm for resolution which is complete for any set of tgds and egds (rather than being complete only for specific subclasses). We then show that a technique of saturation can be used to achieve completeness with respect to First-Order (FO) query rewriting. As an application, we generalize a few existing classes (including (S)) that ensure the existence of a finite FO-rewriting.

We then consider a more general notion of rewriting, called Datalog rewriting, and show that it provides a truly unifying paradigm of tractability for the family of Datalog\(^+\) languages. While the classes (S), (T) and (G) are incomparable, we show that every set of rules in (S), (T) or (G) can be rewritten into an equivalent set of standard Datalog rules.

On the negative side, this means intuitively that Datalog\(^+\) does not extend the expressive power of Datalog in the context of query answering. On the positive side however, one may use the flexible syntax of Datalog\(^+\) while using (only) standard Datalog in the background, thus making use of existing optimization techniques, such as Magic-Set.

1. INTRODUCTION

While the language Datalog and its extensions have been studied for decades (see e.g. [12, 2, 1]), they recently received a renewed attention. In particular Gottlob et al introduced a comprehensive and unifying framework called Datalog\(^+\) ([10, 8, 9, 11]) which is based on a family of Datalog extensions. One of the main qualities of this framework lies in its generality. In particular, Datalog\(^+\) is expressive enough to cover some interesting classes of ontologies, some light-weight description logics and some fragments of F-logic (see e.g. [10]). It was also argued in [10] that Datalog\(^+\) is useful in a variety of contexts, for example Data Exchange [15] and the Semantic Web [2]. The main reason for this, is that Datalog\(^+\), unlike standard Datalog, addresses the fundamental problem of value invention by considering rules with fresh variables in the head. Such rules are known as tuple-generating dependencies (tgds in short) and correspond to first-order formulas of the form

\[
\forall x, y, \phi(x, y) \rightarrow \exists z, \psi(x, z)
\]

where \(\phi\) and \(\psi\) are two conjunctions of atoms (that may express joins) while \(z\) is a tuple of existentially quantified variables that can be used to reason about unknown entities. In addition, Datalog\(^+\) supports equality-generating dependencies (egds in short) of the form

\[
\forall x, y, z, \phi(x, y, z) \rightarrow x = y
\]

where \(\phi\) is a conjunction of atoms. Tgds and egds can be used to encode typical schema dependencies such as inclusion dependencies or function dependencies (see e.g. [1]) which, in turn, allow to reason about structured data. On the negative side, however, the problem of query answering under arbitrary tgds and egds is undecidable [8], and it was observed in [8] that the problem remains undecidable even for a fixed set tgds.

To avoid undecidability, the Datalog\(^+\) framework typically considers (in [10]) three alternative restrictions called termination, guardedness and stickiness, defined as follows:

Termination: There exists a chase procedure [15, 14, 8, 20] that, for a given database, computes a universal solution in polynomial time (data complexity) which, in turn, can be used for sound and complete query answering.

Guardedness: The set of dependencies consists of guarded tgds and separable egds, in which case, as shown in [8], it is possible to reach tractability, while remaining complete, by stopping the oblivious chase after a fixed depth.

Stickiness: The set of dependencies is a set of tgds (and separable egds) ensuring that every chase derivation is sticky [11], meaning intuitively that the fresh variables introduced by the chase are only propagated in an harmless way.
The three classes discussed above are unfortunately incomparable, and the only property that really unifies them in [10] is the fact that they are all based on the chase, either in their definition or in terms of properties. This approach has several advantages and it contributes, in particular, to the simplification of the ‘big picture’. However, there are alternative ways of approaching the above classes. For example, the guarded fragment can be decided, at least in some cases, using tableaux algorithms or resolution algorithms (see e.g. [24, 19, 17, 16]). Similarly, the criterion of stickiness can be understood as a criterion ensuring the termination of resolution (as opposed to a specific property of the chase). In fact, a custom resolution procedure was proposed in [11] for the case of stickiness. However, this procedure relies on the specific properties of sticky tgds, and it is relevant only for this class of dependencies. In contrast with [11], a contribution of this paper will be to consider a resolution algorithm which is defined (and complete) for arbitrary sets of tgds and egds, as to identify larger tractable classes.

While exploring some alternatives to the chase procedure, this paper follows a similar methodology to that of Datalog\textsuperscript{±} in the sense that it aims at identifying a unifying paradigm. Towards this goal, we will consider, beside resolution, several notions of rewriting:

- **Data Rewriting (a.k.a. Universal Solutions).**
  Given a database $D$ and a set of dependencies $\Sigma$, a data rewriting for $D$ is a new database $D'$ which integrates all the information can be inferred from $\Sigma$. Given such a rewriting $D'$, we can then test whether a conjunctive query $Q$ is implied by $D$ and $\Sigma$ by testing whether $Q$ is implied by $D'$. This technique can be used whenever the chase terminates since a universal solution is in fact a special case of data rewriting.

- **Query Rewriting (a.k.a. First-Order Rewriting).**
  Given a query $Q$ and a set of dependencies $\Sigma$, a query rewriting for $(\Sigma, Q)$ is a query $Q_\Sigma$ (in a first-order language) which integrates all the information encoded in $\Sigma$. Given such a rewriting $Q_\Sigma$, we can then test whether a conjunctive query $Q$ is implied by $\Sigma$ and a database $D$ by testing whether $Q$ is implied by $D$. As already observed in [11], the technique of query rewriting can be used whenever $\Sigma$ is a set of sticky tgds.

- **Datalog Rewriting (The Unifying Paradigm).**
  Given a conjunctive query $Q$ and a set of dependencies $\Sigma$, a Datalog rewriting for $(\Sigma, Q)$ consists of a new pair $(\Sigma', Q')$ where $\Sigma'$ is a set of standard Datalog rules (without fresh variables in the head) and $Q'$ is a query such that $(\Sigma', Q')$ is equivalent to $(\Sigma, Q)$ with respect to query answering.

As we will show in this paper, the technique of Datalog rewriting is not only a strictly generalisation of (first-order) query rewriting, but it can also be used in the case of terminating dependencies (instead of data rewriting) and in the case of guarded dependencies (instead of relying on the chase). In this sense, Datalog rewriting is therefore a truly unifying paradigm for the family of Datalog\textsuperscript{±} languages. Also, Datalog rewriting was proved useful for classes of dependencies that are not (yet) covered by the Datalog\textsuperscript{±} framework, in particular in the context of Description Logic (see e.g. [25]). In this sense, heuristics for Datalog rewriting can be used to further generalize the Datalog\textsuperscript{±} framework.

### Structure and Main Contributions

The preliminary section formalizes the problem of query answering (under constraints) as a basic implication problem. For the sake of concision and symmetry, the key notions of databases, queries and dependencies are all based of sets of atoms with variables only (called instances). Constants, null values, and non-boolean queries with equality atoms are discussed in Section 3.5.

1. In Section 3.1, we propose a concise definition of resolution for arbitrary tgds and egds and show that it is complete for query answering. In contrast with alternative approaches (such as [23]), this definition does not rely on skolemization and unskolemization. Instead, it relies simply on instances and renamings (a.k.a. homomorphisms).

2. In Section 3.2, we show that a technique of saturation can be used to reach completeness with respect to finite rewritability. More precisely, the saturated chase (like the core chase of [14]) computes a finite data rewriting whenever there exists one. Similarly, the saturated resolution computes a finite query rewriting whenever there is one.

3. In Section 3.3, we focus on tgds and revisit the notion of stickiness. We observe that there is a reduction from the class of sticky tgds to the class of lossless tgds which, in fact, can be used to identify larger classes or rewritable queries.

4. In Section 3.4, we show that query rewriting under egds (as opposed to tgds only) is more complex. We then propose a heuristic for the integration of a set of egds into a set of tgds, thus allowing (in some cases) to rely on stickiness and query rewriting despite the presence of conflicting egds.

5. In Section 4.1, we show that data rewriting captures the class of dependencies ensuring the termination of the oblivious chase (from [20]), and as a special case, the class of weakly acyclic dependencies (from [15]).

6. In Section 4.2, we finally show that data rewriting also captures the class of guarded tgds. This result is the most technical and arguably the most important contribution.

The proofs (or more accurately, the proof sketches) have been included to the body of the paper. Readers unfamiliar with the problem of query answering under tgds and egds are invited to consult e.g. [10] for more examples, applications and related works. Additional proof details (for some of the proofs) can be found in [21].

### 2. PRELIMINARIES

We chose the infinite model semantic. Recall however that it coincides with the finite model semantic under either termination ([14] [20]), guardedness ([21]), or stickiness ([17]).

#### Instances and Dependencies

Let $V$ be a countable set of variables and let $\sigma$ be a finite set of predicates. We assume that each $R \in \sigma$ comes with a fixed and finite arity $a_R$. An instance $I$ is a set of atoms $R(\bar{v})$ where $R \in \sigma$ and $\bar{v}$ is a tuple
of variables respecting the arity of $R$, that is, $v \in V^n$. We let $V_I$ be the set of variables occurring in an instance $I$. A tgd is a rule $B \rightarrow H$ where $B$ and $H$ are two finite instances called the body and the head of $r$, respectively. We say that a tgd $r$ is of the form $B(X, Y) \rightarrow H(X, Z)$ when $X = V_B \setminus V_H$, $Y = V_Y \setminus X$, and $Z = V_Z \setminus X$. Such a tgd $r$ is called a datalog rule when $Z = \emptyset$. An $en$ is $r$ is a rule $B \rightarrow x = y$ where $B$ is a finite instance and $x, y \in V_B$. An $en$ $r$ is of the form $B(x, y, Z) \rightarrow x = y$ when $Z = V_B \setminus \{x, y\}$.

**Semantics.** A renaming is a mapping from $V$ to $V$. Given an instance $I$ and a renaming $\theta$, we let $I[\theta]$ be the set of atoms $R(\theta(t_1), \ldots, \theta(t_n))$ where $R(t_1, \ldots, t_n) \in I$. Given two instances $I$ and $J$ we let $I \models J$ when $J[\theta] \subseteq I$ for some renaming $\theta$. Given an instance $I$ and a set $J$ of instances, we let $I \models J$ when $I \models J$ for some $J \in J$. Given two sets $I$ and $J$ of instances, we let $I \equiv J$ when $I \models J$ for all $I \in I$. Note that these definitions are consistent with respect to singletons. In particular, we have $I \models J \iff \{I\} \models \{J\}$. Given two sets of instances $I$ and $J$ we write $I \equiv J$ when both $I \models J$ and $J \models I$.

Given $X \subseteq V$, a renaming of $X$ is a renaming $\theta$ such that $\theta(v) = v$ for all $v \in (V \setminus X)$. In the following, we use the notation $\theta|_X$ to indicate that $\theta|_X$ is a renaming of $X$. Given $Y \subseteq V$ disjoint from $X$, and a renaming $\theta|_Y$, we denote by $\theta|_{X \cup Y}$ the renaming of $X \cup Y$ that coincides with $\theta|_X$ on $X$ and with $\theta|_Y$ on $Y$.

Given an instance $I$ and a tgd $r : B(X, Y) \rightarrow H(X, Z)$, we let $I \models r$ when for all $\theta|_X$ and $\theta|_Y$ such that $B[\theta|_X, \theta|_Y] \subseteq I$, there exists $\theta|_Z$ such that $H[\theta|_X, \theta|_Z] \subseteq I$. Given an instance $I$ and an egd $r : B(x, y, z) \rightarrow x = y$, we let $I \models r$ when for all $\theta|_x$, $\theta|_y$, and $\theta|_z$ such that $B[\theta|_x, \theta|_y, \theta|_z] \subseteq I$, it holds that $\theta|_x(x) = \theta|_y(y)$. Given a set $\Sigma$ of tgds and egds, we finally write $I \models \Sigma$ when $I \models r$ for all tgds and egds $r \in \Sigma$.

Note that, for two instances $I$ and $J$, the property $I \models J$ corresponds to several equivalent intuitions: the boolean conjunctive query $J$ is true in the database $I$; the query $J$ is contained in the query $I$; the formula $J[I]$ is implied by $I$; the instance $I$ is a model of $J$; or there is a homomorphism from $J$ to $I$. A similar comment holds for sets of instances.

In the following definition, $D$ and $Q$ denote two sets of instances and $\Sigma$ denotes a set of dependencies. Intuitively, $D$ corresponds to a database or a set of databases (representing a set of possible models $M$). $\Sigma$ corresponds to an ontology or a set of structural dependencies; and $Q$ corresponds to a union of boolean conjunctive queries.

**Definition 1.** We say that $Q$ is certain in $\Gamma$ under $\Sigma$, denoted $D \upmodels \Sigma \models Q$, if, for all instance $M$ such that $M \models D$ and $M \models \Sigma$, we have $M \models Q$.

**Chase.** We next recall the definition of the chase [15] [14] [8] [20]. More precisely, the following definition coincides with [8] in the case of tgds and with [15] in the case of egds.

Given an instance $I$ and a set of variables $Z$, we say that $\theta_Z$ is an $I$-fresh renaming when $\theta_Z$ is a renaming of $Z$, $\theta_Z$ is injective on $Z$ and its image $\theta_Z(Z)$ is disjoint from $V_I$. Given two variables $u$ and $v$ we denote by $[u \mapsto v]$ the unique renaming $\theta$ of $\{v\}$ satisfying $\theta(v) = u$. Given an instance $I$, and a set $\Sigma$ of tgds and egds, we write $I^{\text{chase}}_\Sigma J$ when either $J = I$ or one of the following rules applies:

(tgd) There is a tgd $r : B(X, Y) \rightarrow H(X, Z)$ in $\Sigma$ and some renumberings $\theta_X$ and $\theta_Y$ such that $B[\theta_X, \theta_Y] \subseteq I$ and $J = I \cup \{H[\theta_X, \theta_Z] \mid \theta_Z \in \Sigma\}$ for some $I$-fresh renaming $\theta_Z$.

(egd) There is an egd $r : B(x, y, Z) \rightarrow x = y$ in $\Sigma$ and some renumberings $\theta_x$, $\theta_y$, and $\theta_Z$ such that $B[\theta_x, \theta_y, \theta_Z] \subseteq I$ and $J = I \cup \{H[\theta_y(y) \mapsto \theta_x(x)]\}$.

Using the symbol $*$ for the transitive closure, we finally write $J \in \text{Chase}(I, \Sigma)$ when $I^{\text{chase}}_\Sigma J$.

**3. FIRST-ORDER RESOLUTION**

**3.1 Definition and Completeness**

We next propose a definition of resolution for tgds and egds which, unlike alternative approaches (such as [24]), does not require any complex algorithm of (un)skolemization.

Given an instance $Q$ and a set $\Sigma$ of tgds and egds, we write $Q^{\text{res}}_\Sigma R$ when one of the following rules applies:

(ren) $R = Q[\theta]$ for some renaming $\theta$.

(tgd) There is a tgd $B(X, Y) \rightarrow H(X, Z)$ in $\Sigma$ and three renumberings $\theta_X$, $\theta_Y$, and $\theta_Z$ such that $R = (Q[\theta_X \mapsto \theta_Y]), B[\theta_X, \theta_Y]$ and the following conditions hold:
- $\theta_Z$ is an injection from $Z$ to $(V \setminus V(R))$.

(egd) There is an egd $B(x, y, Z) \rightarrow x = y$ in $\Sigma$ and three substitutions $\theta_x$, $\theta_y$, and $\theta_Z$ such that $B[\theta_x, \theta_y, \theta_Z] \subseteq I$, it holds that $\theta_x(x) = \theta_y(y)$. Given a set $\Sigma$ of tgds and egds, we finally write $I \models \Sigma$ when $I \models r$ for all tgds and egds $r \in \Sigma$.

We finally write $R \in \text{Resol}(Q, \Sigma)$ when $Q^{\text{res}}_\Sigma R$.

**Theorem 1.** For all set $\Sigma$ of tgds and egds, and for all instances $D$ and $Q$, the following statements are equivalent:

1. $D \land \Sigma \models Q$
2. $\exists U \in \text{Chase}(I, \Sigma), U \models Q$
3. $\exists R \in \text{Resol}(Q, \Sigma), D \models R$

**Proof Sketch.** The equivalence between (1) and (2) is well-known. Recall however that it holds only under the infinite model semantics (see e.g. [14] for more details). The implication (3) $\Rightarrow$ (1) means that the resolution is sound, and it is fairly straightforward. We next prove that (2) implies (3). We proceed by induction, and show that the following property holds for all $n \geq 0$:

If there is a chase derivation of length $n$ of the form $D = I_0 \upmodels_{\Sigma_0} I_1 \upmodels_{\Sigma_1} \ldots \upmodels_{\Sigma_n} I_n$ where $I_n \models Q$ and there exists $R \in \text{Resol}(Q, \Sigma)$ such that $D \models R$.

In the case $n = 0$, the property holds with $R = Q$. Assume now the property true for $n - 1$ and consider a derivation of length $n$ as above. Let $h$ be a renaming such that $Q[h] \subseteq I_n$ and let $Q' = Q[h]$. Because of the resolution rule (ren), we have $Q^{\text{res}}_\Sigma Q'$. We now distinguish two cases:
(1) Assume that $I_o$ is obtained by chasing a tgd
\[ B(X, Y) \rightarrow H(X, Z) \]
and consider $\theta_X, \theta_Y, \theta_Z$ such that $B[\theta_X, \theta_Y] \subseteq I_{o-1}$ and $I_o = I_{o-1} \cup H[\theta_X, \theta_Z]$. If $Q' \subseteq I_{o-1}$, we can apply the induction hypothesis for $n - 1$. Otherwise, there is at least one atom in $Q' \cap H[\theta_X, \theta_Z]$ and we have $Q' \rightarrow_{\text{chase}} R$ for
\[ R = (Q' \setminus H[\theta_X, \theta_Z]) \cup B[\theta_X, \theta_Y]. \]
Since $R \subseteq I_{o-1}$ we have $I_{o-1} \models R$ and we can easily conclude the inductive step.

(2) Assume that $I_o$ is obtained by chasing an egd
\[ B(x, y, Z) \rightarrow x = y \]
and consider $(\theta_x, \theta_y, \theta_Z)$ such that $B[\theta_x, \theta_y, \theta_Z] \subseteq I_{o-1}$ and $I_o = I_{o-1}[\theta]'$ for $\theta' = [\theta(y) \leftarrow \theta_x(x)]$. For each atom $A \in Q$, choose an atom $c(A) \in I_{o-1}$ such that $\theta'(c(A)) = A$ and let $R = \{ c(A), A \in Q' \} \cup B[\theta_x, \theta_y, \theta_Z]$. We can finally check that $Q' \rightarrow_{\text{res}} R$ and conclude the inductive step.

### 3.2 Rewriting and Saturation

We next formalize the notions of data rewriting and query rewriting before showing that completeness (with respect to finite rewritability) can be reached, in both cases, by saturating the relation $\rightarrow_{\text{chase}}$ or $\rightarrow_{\text{res}}$ in a rather natural way.

**Definition 2.** Consider a set $\Sigma$ of dependencies and two sets $\mathcal{D}$ and $\mathcal{Q}$ of instances. A data rewriting for $(\mathcal{D}, \Sigma)$ is a set of instances $\mathcal{U}$ such that, for all set $Q'$ of instances:
\[
(\mathcal{D} \wedge \Sigma \models Q') \text{ iff } (\mathcal{U} \models Q').
\]
Similarly, a query rewriting for $(\Sigma, \mathcal{Q})$ is a set of instances $\mathcal{R}$ such that, for all set $Q'$ of instances:
\[
(\mathcal{D}' \wedge \Sigma \models Q') \text{ iff } (\mathcal{R} \models Q').
\]

**Saturation.** Let $\rightarrow \in \{ \rightarrow_{\text{chase}}, \rightarrow_{\text{res}} \}$. Given two finite sets of instances $\mathcal{I}$ and $\mathcal{J}$, we write $\mathcal{I} \rightarrow \mathcal{J}$ when, intuitively, $\mathcal{J}$ is a concise representation (up to equivalence) of all the instances $J$ such that $\mathcal{I} \rightarrow J$ for some $J \in \mathcal{I}$. More formally, we let $\mathcal{I} \rightarrow \mathcal{J}$ when $\mathcal{J}$ can be returned by the following non-deterministic algorithm:

- **Start with** $J := I$.
- **Repeat** (until fixed point)
  - If there is an instance $J$ such that:
    1. $I \rightarrow_{\Sigma} J$ for some $I \in \mathcal{I}$; and
    2. $I$ are in the case $\rightarrow = \rightarrow_{\text{chase}}$ and there is no $J' \in \mathcal{J}$ such that $J = J'$; or
    3. $I$ are in the case $\rightarrow = \rightarrow_{\text{res}}$ and there is no $J' \in \mathcal{J}$ such that $J' \models J$.
  - Then let $J := J \cup \{ J \}$
  - Return $J$.

Note that the above definition can easily be translated into an actual algorithm since the set of instances $I$ satisfying the point (1) is finite up to isomorphism (and can be enumerated in exponential time). We define a *complete derivation* for $\mathcal{I}$ and $\rightarrow$ as an infinite series $(\mathcal{I}_i)_{i \geq 0}$ where $\mathcal{I}_0 = \mathcal{I}$ and $\mathcal{I}_0 \rightarrow \mathcal{I}_{i+1}$ for each $i \geq 0$. We finally let $\mathcal{J} \in \text{Fix}(\mathcal{I}, \rightarrow)$ when $\mathcal{J} = \bigcup_i \mathcal{I}_i$ for some complete derivation $(\mathcal{I}_i)_{i \geq 0}$.

**Theorem 2.** For all set $\Sigma$ of tgds and egds, and for all sets $\mathcal{D}$ and $\mathcal{Q}$ of instances:
- Every $\mathcal{U} \in \text{Fix}(\mathcal{D}, \rightarrow_{\text{chase}})$ is a data rewriting of $(\mathcal{D}, \Sigma)$.
- Every $\mathcal{R} \in \text{Fix}(\mathcal{Q}, \rightarrow_{\text{res}})$ is a query rewriting of $(\Sigma, \mathcal{Q})$.

**Proof Sketch.** The result would follow directly from Theorem 1 if we had removed the requirements (2.1) and (2.2) in the above definition of saturation. To prove that the result still holds with (2.1) and (2.2), it is enough to observe that the chase and the resolution are both monotone:

- If $J \models J'$ and $J' \rightarrow_{\text{chase}} K'$, then there exists an instance $K$ such that $J \rightarrow_{\Sigma} K$ and $K \models K'$.
- If $J \models J'$ and $J' \rightarrow_{\text{res}} K'$, then there exists an instance $K$ such that $J \rightarrow_{\Sigma} K$ and $K \models K'$.

The monotonicity of the chase is well-known (see e.g. [20]). For the monotonicity of the resolution, consider $J \models J' = J$ and $J' \rightarrow_{\text{res}} K'$. The case where $K'$ is obtained from $J'$ with the resolution rule (ren) is straightforward. Consider now the case (tgd) and unfold the definition of resolution so that $K' = (J' \setminus H[\theta_X, \theta_Z]) \cup B[\theta_X, \theta_Y]$.

Let $h$ be a renaming of $V_J$ such that $J[h] \subseteq J'$. If $J[h]$ does not intersect $H[\theta_X, \theta_Z]$ we have $J[h] \subseteq K'$ and the property holds for $K = K'$. Otherwise, considering the instance $K = ((J[h]) \setminus H[\theta_X, \theta_Z]) \cup B[\theta_X, \theta_Y]$.

We can check that $I \rightarrow_{\Sigma} I[h] \rightarrow_{\Sigma} K$ and $K[h] \subseteq K'$. Consider finally the case (egd) and unfold the definition of resolution for an egd $B(x, y, Z) \rightarrow x = y$ of $\Sigma$ and three substitutions $\theta_x, \theta_y, \theta_Z$. For each atom $A \in J'$, let $c(A)$ be the atom of $K'$ that has been obtained from $A$ by renaming some occurrences of $\theta_x(x)$ into $\theta_y(y)$, and observe that:
\[
K' = \{ c(A) \mid A \in J' \} \cup B[\theta_x, \theta_y, \theta_Z].
\]

Let $h$ be a renaming of $V_J$ such that $J[h] \subseteq J'$ and let $K = \{ c(A) \mid A \in J[h] \} \cup B[\theta_x, \theta_y, \theta_Z]$.

As in the previous case, we can finally check that we have $I \rightarrow_{\Sigma} I[h] \rightarrow_{\Sigma} K$ and $K[h] \subseteq K'$. This concludes the proof of monotonicity and the proof of Theorem 2.

**Termination.** Let $(\mathcal{I}_i)_{i \geq 0}$ be a complete derivation of $\mathcal{I}$ and $\rightarrow$. We say that this derivation terminates iff there exists a finite $i$ such that $I_{i+1} = I_i$. Note that, in this case, we also have $I_i = I_i$ for all $i \geq 1$. We finally say that $\text{Fix}(\mathcal{I}, \rightarrow_{\Sigma})$ is finite when all the instances $J \in \text{Fix}(\mathcal{I}, \rightarrow_{\Sigma})$ are finite, meaning (equivalently) that all the derivations of $\mathcal{I}$ by $\rightarrow_{\Sigma}$ are finite.

**Theorem 3.** Given a finite set $\Sigma$ of tgds and two finite sets $\mathcal{D}$ and $\mathcal{Q}$ of instances, the following statements hold:
- If there exists a finite data rewriting for $(\mathcal{D}, \Sigma)$, then $\text{Fix}(\mathcal{D}, \rightarrow_{\Sigma})$ is finite.
- If there exists a finite query rewriting for $(\Sigma, \mathcal{Q})$, then $\text{Fix}(\mathcal{Q}, \rightarrow_{\Sigma})$ is finite.
Proof Sketch. The two points are similar and we prove here the second one. Suppose that there exists a finite query rewriting \( \mathcal{R} \) for \((\Sigma, Q)\) and consider a complete derivation \((\mathcal{I}_i)_{i \geq 0}\) for \(Q\) and \(\text{rew}\). Since \(\mathcal{R}\) and \(\bigcup \mathcal{I}_i\) are two rewritings of \((\Sigma, Q)\), we have \(\mathcal{R} \equiv \bigcup \mathcal{I}_i\). Since \(\mathcal{R}\) is finite, there exists a finite \(k\) such that \(\mathcal{R} \equiv \mathcal{I}_k\). We can then observe that \(\mathcal{I}_k = \mathcal{I}_{k+1}\) and conclude that \((\mathcal{I}_i)_{i \geq 0}\) is finite.

Note that a direct consequence of Theorems 2 and 3 is the following: as soon as there is a consequence, every (complete) derivation terminates and new simplifications may therefore be possible in \(\Sigma_A, r\). In fact, the results stated in Section 3.1 can be used to ‘simplify’ the big picture on query rewritability. In particular, we first show that it provides a concise proof of finite query-rewritability of a few well-known classes of dependencies. In fact, the results stated in the following proposition are rather well-known. It is however interesting to compare the following proof sketch (based on resolution, and arguably simple) with, for example, the seminal paper of Johnson and Klug [18] where a proof was given (based on the chase, and arguably complex) for the tractability of inclusion dependencies. This class of dependencies is indeed covered (strictly) by the class of local-as-view tgds (lav tgds) defined in the following proposition.

Proposition 1. In each of the following cases, we can effectively compute a finite query rewriting for \((\Sigma, Q)\):

- \((\text{Lav tgds})\) \(\Sigma\) is a set of tgds \(B \to H\) where the body \(B\) contains at most one atom.
- \((\text{Lossless tgds})\) \(\Sigma\) is a set of tgds \(B \to H\) where, for each atom \(A_r \in H\), we have \(V_B \subseteq V_{A_r}\).
- \((\text{Acyclicity})\) \(\Sigma\) is a set of tgds and there is a linear order \(\leq\) on the predicates of \(\Sigma\) such that, for all tgd \(B \to H\) in \(\Sigma\), all predicate \(R_0\) occurring in \(B\), and all predicate \(R_k\) occurring in \(H\), we have \(R_0 \leq R_k\).

Proof Sketch. In the case of lav tgds, we can observe that the resolution rules (hom) and (tgd) never increase the number of atoms. More formally, whenever \(I \overset{\text{rew}}{\to} J\), we have \(|J| = |I|\). There are therefore a finite number of instances (up to \(\equiv\)) that can be computed by resolution. As a consequence, every (complete) derivation terminates and every \(\mathcal{R} \in \text{Fix}(D, \overset{\text{rew}}{\to})\) is a finite query rewriting. In the case of lossless tgds, the result follows from a similar observation: the resolution rules (hom) and (tgd) never increase the number of variables. (That is, \(I \overset{\text{rew}}{\to} J\) implies \(V_J \subseteq V_I\)). Consider finally the case of an acyclic set \(\Sigma\) of tgds. Let \(\Sigma_0\) be the set of predicates occurring in \(\Sigma\) and consider the ordering \(\Sigma_0 = \{R_0, \ldots, R_n\}\) where \(R_{i+1} \leq \Sigma R_i\) for each \(i \leq n\). For each instance \(I\), consider \(s(I) = (a_1, \ldots, a_n)\) where, for each \(i \in \{1, \ldots, n\}\), \(a_i\) is the number of atoms \(R_i(\bar{v})\) in \(I\) where \(R_i = R_i\). We can observe that, whenever \(I \overset{\text{rew}}{\to} J\), the tuple \(s(J)\) is smaller than \(s(I)\) with respect to lexicographic order. We can finally conclude as in the previous cases.

We next shed more light on the notion of stickiness from [19] which was discussed in the introduction. In particular, we show that there is in fact a direct reduction (preserving the property of finite rewritability) from the class of sticky tgds to the class of lossless tgds. This reduction proves useful in two ways: (1) it provides a direct proof of rewritability for the class of sticky tgds (which, unlike [19], does not require a custom resolution algorithm), (2) and it also allows us to identify a more general class of rewritable dependencies. In a nutshell, the key idea of the following reduction consists in replacing an atom \(A(x, y)\) by an atomic formula \(R(\bar{x}) \approx (\exists y, A(x, \bar{y}))\) whenever \(A(x, \bar{y})\) is the only atom (in a given tgd) where the variables of \(\bar{y}\) occur.

Simplifying atoms. Given a tgd \(r : B \to H\) and an atom \(A\) in the body \(B\), we let \(X_{A, r} = V_A \cap V_B\) and \(Y_{A, r} = V_A \setminus X_{A, r}\). We then say that \(A\) can be simplified in \(r\) when \(Y_{A, r}\) is non-empty and disjoint from \(V_{B \setminus r}\). For example, in the tgd

\[
A(x_1, y_1), B(x_1, x_2, y_2), C(x_1, y_3), D(y_2) \to R(x_1, x_2, z_1)
\]

the atoms \(A(x_1, y_1)\) and \(C(x_1, y_3)\) can be simplified. In contrast, \(B(x_1, x_2, y_2)\) cannot be simplified because \(y_2\) occurs in an other atom of the body.

Consider now a set \(\Sigma\) of tgds, a tgd \(r : B \to H\) in \(\Sigma\), and an atom \(A\) which can be simplified in \(r\). Given a tgd \(r' : B' \to H'\) in \(\Sigma\) and an atom \(A' \in H'\) we say that \(r'\) unifies with \((A, r)\) when there exists a renaming \(\theta_1\) of \(V_A\) and a renaming \(\theta_2\) of \(V_{B'} \cap V_{H'}\) such that \(A[\theta_1] \in H'[\theta_2]\), in which case \((\theta_1, \theta_2)\) is called a unifier of \(r'\) and \((A, r)\). The following algorithm describes the new set of tgds \(\Sigma_{A, r}\) that results from the simplification of \((A, r)\) in \(\Sigma\):

- Let \(\bar{x} = (x_1, \ldots, x_n)\) be an ordering of \(X_{A, r}\).
- Let \(\bar{R}_0\) be a fresh predicate of arity \(n\).
- Replace \(A\) by \(\bar{R}_0(\bar{x})\) in the body of \(r\).
- For all unifier \((\theta_1, \theta_2)\) of \(r'\) and \((A, r)\), add the tgd \(B'[\theta_2] \to R_0(\theta_1(\bar{x}))\).

Note that the quantification of the variables may be modified in this process, and new simplifications may therefore be possible in \(\Sigma_{A, r}\). For instance, the set of tgds

\[
\begin{align*}
\text{r}_1 & : A(x, x, y, z, t) \to B(x, y) \\
\text{r}_2 & : C(x, y) \to \exists u, v, A(x, y, u, v, v) \\
\text{r}_3 & : D(x, y, z, t) \to A(x, x, y, z, t)
\end{align*}
\]

will, after a simplification step in \(r_1\), be replaced by

\[
\begin{align*}
\text{r}'_1 & : \bar{R}_0(x, y) \to B(x, y) \\
\text{r}_2 & : C(x, y) \to \exists u, v, A(x, y, u, v, v) \\
\text{r}_3 & : D(x, y, z, t) \to A(x, x, y, z, t) \\
\text{r}'_2 & : C(x, x) \to \exists u, \bar{R}_0(x, u) \\
\text{r}_3 & : D(x, y, z, t) \to \bar{R}_0(x, y)
\end{align*}
\]

and the first atom of \(r'_2\) can now be simplified (even though this atom could not be simplified in \(r_3\)).

Despite the previous observation, we can check that the process of simplification always terminates since each step introduces only a finite number of tgds, and the number of variables in each of these tgds is strictly decreasing. For a finite set \(\Sigma\) of tgds, we finally define (non-deterministically) the set \(\Sigma_{\downarrow}\) as a set of tgds obtained from \(\Sigma\) by repeating the operation of simplification until a fixed point is reached.
Theorem 4. For all finite sets \( \Sigma \) of tgds and all finite sets \( Q \) of instances, there is a finite query rewriting for \((\Sigma, Q)\) iff there is a finite query rewriting for \((\Sigma_\downarrow, Q)\).

Proof Sketch. Consider a series of simplification steps \( s_1, \ldots, s_n \), where each \( s_i \) is characterized by the atom \( A_i(\bar{x}_i, \bar{y}_i) \) that has been simplified at step \( s_i \) and the corresponding atom \( R_i(\bar{z}_i) \) that has been introduced. For each \( i \), consider the tgd \( r_i : A_i(\bar{x}_i, \bar{y}_i) \rightarrow R_i(\bar{z}_i) \). Finally, let \( \Gamma = \{ r_i \}_{i \in \{1, \ldots, n\}} \). We can observe that, for all instance \( D \) over the original schema, and every data rewriting \( D_f \) for \((\Gamma, \Gamma)\), we have \( D \wedge \Sigma \models \exists t \forall \exists \frac{\Gamma}{Q} \) iff \( D_f \wedge \Sigma \downarrow \models \exists t \forall \exists \frac{\Gamma}{Q} \). Since, \( \Gamma \) is a set of Datalog rules, there is a data rewriting \( D_f \) which is finite iff \( D \) is finite. We can also observe that if the set of tgds \( \Sigma^{-1} = \{ (\bar{r}_i \downarrow)_{i \in \{1, \ldots, n\}} \} \) exists a one-to-one correspondence (which preserves finiteness) of the extended schema on the original schema, we can then observe that, for all instance \( D' \) of the original schema, there is therefore a data rewriting \( D'_f \) of \((\Sigma^{-1}, \Sigma)\) which is finite iff \( D' \) is finite.

With letting \( \Pi \) be the operation that projects an instance of the extended schema on the original schema, we can then observe that, for all instance \( D \) of the original schema, we have \( \Pi((D_f)_{\downarrow^{-1}}) \equiv D \). This means intuitively that there exists a one-to-one correspondence (which preserves finiteness) between the instances of the original schema and the instances of the extended schema. This is the key argument behind the proof of Theorem 4.

Stickiness (Slightly Revisited). Given a set of atoms \( B \) and a term \( t \), we denote by \( \text{pos}(t, B) \) the set of pairs \( (r, i) \) of \( (R, i) \) the set of pairs \( (r, i) \) such that \( B \) contains an atom \( R(t_1, \ldots, t_n) \) where \( t_i = t \). Given a set of tgds \( \Sigma \), a tgd \( B \rightarrow H \) in \( \Sigma \) and an atom \( A \in H \), the tgd \( B \rightarrow A \) is called a global-as-view projection of \( \Sigma \), denoted \( r' \in \text{Gav}(\Sigma) \). Given a set of tgds \( \Sigma \), we define the set \( \text{A}_{\Sigma} \) of affected positions as the smallest set of positions such that, for all tgd \( r \in \text{Gav}(\Sigma) \) of the form \( r : B(X,Y) \rightarrow H(X,Z) \), we have:

(i) \( \forall \bar{v} \in Y, pos(v, B) \subseteq \text{A}_{\Sigma} \)

(ii) \( \forall u \in X, (pos(u, H) \subseteq \text{A}_{\Sigma}) \Rightarrow (pos(u, B) \subseteq \text{A}_{\Sigma}) \)

We say that \( \Sigma \) is stickly iff, for all tgd \( r \in \text{Gav}(\Sigma) \) of the form above, and all \( u \in X \) such that \( pos(u, H) \subseteq \text{A}_{\Sigma} \), the variable \( u \) occurs in only one atom of the body. This definition differs from \([11]\) because of this last requirement “in only one atom”. In contrast, the definition from \([11]\) requires that \( u \) occurs “only once” (in only one atom and in only one position).

Theorem 5. If \( \Sigma \) is a sticky set of tgds, then \( \Sigma_\downarrow \) is a set of lossless tgds, and therefore, for all set \( Q \) of instances, there is a finite query rewriting for \((\Sigma, Q)\).

Proof Sketch. It can be checked that, for every sticky set \( \Sigma \) of tgds, and every atom \( A \) that can be simplified in a tgd \( r \in \Sigma \), the set \( \Sigma_\downarrow \) resulting from the simplification of \((A, r)\) is also a sticky set of tgds. Assume now that \( \Sigma_\downarrow \) is sticky and \( \Sigma_\downarrow \) contains a tgd \( r : B \rightarrow H \) which is not lossless. Since \( r \) is not lossless, there is an atom \( A_h \in H \) and an atom \( \bar{A}_B \) in \( B \) such that \( \forall A_h \not\subseteq \forall A_h ' \). Observe that the tgd \( B \rightarrow A_h \) belongs to \( \text{Gav}(\Sigma_\downarrow) \) and consider a variable \( v \in \forall A_h \setminus \forall A_h ' \). By definition of \( \text{A}_{\Sigma_\downarrow} \), this variable \( v \) occurs in an affected position, and the stickiness assumption ensures that \( v \) occurs only in the atom \( A_h \), meaning that \( v \not\subseteq \forall A_h \). If follows that \( A_h \) can be simplified in \( r \), and this contradicts the definition of \( \Sigma_\downarrow \). Therefore, every tgd in \( \Sigma_\downarrow \) is lossless.

We observe that the (revisited) notion of stickiness is simultaneously a strict generalisation of: (1) the class of lossless tgds; (2) the original notion defined in \([11]\) from which it is inspired; and (3) the class of vac tgds, which was not yet covered by (2). Note also that stickiness could be combined with the (incomparable) notion of acyclicity discussed in Proposition \([1]\) and/or the class of sticky-join tgds introduced in \([11]\) to design an ever larger class of tractable settings. However, this is left as future work.

3.4 Integrating the Egds

This section provides a negative result on egds and query rewriting which will motivate two further contributions: (1) a novel technique, also in this section, that allows the integrating of some egds in a set of tgds; and (2) the study, in Section 4, of a richer notion of rewriting based on Datalog.

While the completeness result from Section 3.1 remains of clear interest with both tgds and egds, it turns out that the notion of query rewriting from Section 3.2 is in fact very limited under egds. Intuitively, this is because we considered a notion of first-order rewriting, while dealing with egds often requires the power of second-order (or, as we will see, the use of some integration technique which extends the schema). In fact, as illustrated by the following proposition, \((\Sigma, Q)\) is rarely rewritable under egds, even in the case where \( \Sigma \) consists of a single egd.

Proposition 2. There is no finite query rewriting for

\[ \Sigma = \{ A(x, y), A(x, y') \Rightarrow y = y' \} \]

and \( Q = \{ R(z, z) \} \).

Proof Sketch. An infinite rewriting for \((\Sigma, Q)\) is the set of instances \( R = \{ R_i \}_{i \geq 2} \) where each \( R_i \) is equal to \( \{ R(x_1, x_n) \cup A(x_1, y_i), A(x_{i+1}, y_{i+1}) \mid i \leq n - 1 \} \). We can check that this rewriting \( R \) is not equivalent (up to \( \equiv \)) to any finite set of instances. Therefore, there is no finite rewriting for \((\Sigma, Q)\).

Despite the above result, it has been observed in \([22]\) that there are practical scenarios where egds can be ‘handled’ with a first order language. It is indeed possible, in some cases, to integrate these egds in the given set of tgds, as to compute a new set of tgds which, intuitively, does not interact with these egds. As a special case, this approach based on integration, covers the scenarios where the given sets of tgds and egds are already non-conflicting, as defined in \([18]\) or \([19]\). However, we will also capture scenarios where the original set of tgds properly interacts with the egds.

As in \([22]\) or \([9]\), we next focus on functional dependencies rather than arbitrary egds. The reason for this is that the egds used in practice often consist of functional dependencies, and the functional dependencies have a more specific syntax which proves more convenient (in the context of integration). Recall that a functional dependency is a rule of the form \( R[a] \rightarrow \{ \forall \alpha \} \rightarrow I_a \) where \( R[a] \) is a predicate of arity \( a \), \( K_a \subseteq \{ 1, \ldots, a \} \) and \( I_a \subseteq \{ 1, \ldots, a \} \setminus K_a \). Given an instance \( M \), we then let \( M \models \alpha \) when, for all atoms of the form \( R(x_1, \ldots, x_n) \) and \( R(x'_1, \ldots, x'_n) \) in \( M \), there either exists some \( k \in K_a \) such that \( x'_k \neq x_k \), or it holds that \( x'_k = x_k \). It is clear that a functional dependency can always be expressed by an equivalent egd. For example, for a binary predicate \( A \), the dependency \( \alpha : A[1] \rightarrow 2 \) is equivalent to the egd \( A(x, y), A(x, y') \Rightarrow y = y' \).

Integration Heuristic. Given a set \( \Sigma \) of tgds and a functional dependency \( \alpha : R[\alpha] \rightarrow I_\alpha \), we let \( \Sigma_\alpha \) be the set of tgds obtained as follows:
Start with $\Sigma_0 := \Sigma$
Let $D_0$ and $F_o$ be two fresh predicates
Let $t_1, \ldots, t_n$ be an ordering of $K_0$
Add to $\Sigma_0$ the two following tgd:

$R_o(x_1, \ldots, x_{a_o}) \rightarrow F_o(x_1, \ldots, x_{i_o}, x_{a_o})$
$D_o(x_1, \ldots, x_{r_o}) \rightarrow \exists y, F_o(x_1, \ldots, x_{r_o}, y)$

For all tgd $r : B \rightarrow H$ in $\Sigma$
For all atom $R_o(t_1, \ldots, t_{a_o})$ in $H$ (\footnote{Note in particular that there is a tgd $R_o$ in the above algorithm are actually used. That is, when $r:B \rightarrow H$ in $\Sigma$ and an atom $A \in H$ of the form $R_o(t_1, \ldots, t_{a_o})$ such that $\{t_1, t_2, \ldots, t_{a_o}\} \subseteq \cal{V}_B$ and $t_{a_o} \notin \cal{V}_B$. Note in particular that $\alpha$ does not interact with $\Sigma$ when $\alpha$ is non-conflicting with $\Sigma$ according to the definition of \cite{9} (which would requires here that $\{t_1, t_2, \ldots, t_{a_o}\} \subseteq \cal{V}_B$.})
\begin{itemize}
\item Add the atom $F_o(t_1, \ldots, t_{a_o})$ to the body of $r$
\item Add the tgd $B \rightarrow D_o(t_1, \ldots, t_{a_o})$ to $\Sigma_o$.\end{itemize}

We say that $\alpha$ interacts with $\Sigma$ when the lines below (\footnote{Note finally that, since $F$ is a set of functional dependencies, we can compute a data rewriting $D_F$ for $(D, F)$ in polynomial time (data complexity) using any standard chase procedure. Combining Theorem \ref{thm:3} with the results of the previous section, we finally get the following result:}) in the above algorithm are actually used. That is, when there is a tgd $r:B \rightarrow H$ in $\Sigma$ and an atom $A \in H$ of the form $R_o(t_1, \ldots, t_{a_o})$ such that $\{t_1, t_2, \ldots, t_{a_o}\} \subseteq \cal{V}_B$ and $t_{a_o} \notin \cal{V}_B$.

\begin{itemize}
\item $D \land \Sigma \land \alpha \models Q$
\item $D \land \Sigma_a \models Q$
\end{itemize}

\textbf{Proof Sketch.} Let $U \in \text{Fix}(D, \Sigma)$ and recall from Theorem 2 that $U$ is a data rewriting for $(D, \Sigma)$. Under the assumptions that $D \models \alpha$ and $\alpha$ does not interact with $\Sigma$, we can check that $U \models \alpha$. It follows that $U$ is also a data rewriting for $(D, \Sigma \land \alpha)$. We can finally check that the following statements are all equivalent: $D \land \Sigma \land \alpha \models Q$; $U \models Q$; $D \land \Sigma \land \alpha \models Q$; and $D \land \Sigma \land \alpha \models Q$.\qed

Consider now a set of functional dependencies $F$ and a set $\Sigma$ of tgd. We say a set of tgd $\Sigma_F$ of tgd integrates $F$ in $\Sigma$ iff there exists a series $\alpha_1, \ldots, \alpha_n$ in $\cal{F}$ and a series $\Sigma_0$, $\Sigma_1$, $\ldots$, $\Sigma_n$ such that:
\begin{itemize}
\item $\Sigma_0 = \Sigma$, $\forall i \Sigma_{i+1} = (\Sigma_i)_{\alpha_i}$ and $\Sigma_n = \Sigma_F$;
\item for all $i$, the integration of $\alpha_i$ succeeds in $\Sigma_i$; and
\item there is no remaining $\alpha \in \cal{F}$ that interacts with $\Sigma_F$.
\end{itemize}

We are now ready to formalize the property of interest which is ensured by the integration heuristic.

\textbf{Theorem 6.} Given a set of tgd $\Sigma_F$ that integrates a set of functional dependencies $F$ in a set of tgd $\Sigma$, for all instances $D$, all data rewriting $D_F$ of $(D, F)$, and all sets $Q$ of instances, the following statements are equivalent:
\begin{itemize}
\item $D \land \Sigma \land \alpha \models Q$
\item $D_F \land \Sigma_F \models Q$
\end{itemize}

\textbf{Proof Sketch.} The result can be proven by induction on the cardinality of $F$ using Lemma \ref{lem:1} and the result of separability which was established in \cite{8}.\qed

\noindent Note finally that, since $F$ is a set of functional dependencies, we can compute a data rewriting $D_F$ for $(D, F)$ in polynomial time (data complexity) using any standard chase procedure. Combining Theorem \ref{thm:3} with the results of the previous section, we finally get the following result:

\textbf{Corollary 1.} Given $\Sigma_F$ that integrates $F$ in $\Sigma$, if the set $\Sigma_F$ is sticky, for all set $Q$ of instances, the following problem is in $\text{PTime}$: given an instance $D$, does $D \land \Sigma \land F \models Q$?

We finally provide an example of scenario taken from \cite{22} which is covered by the approach described in this section:
\[
\begin{align*}
\Sigma &= \{A(x, y) \rightarrow \exists z, B(x, z) \land C(z, y)\} \\
F &= \{\alpha : B[1] \rightarrow 2\} \\
\Sigma_F &= \left\{B(x, y) \rightarrow F_o(x, y), D(x) \rightarrow \exists y, F_o(x, y), A(x, y) \rightarrow D_o(x), A(x, y) \land F_o(x, z) \rightarrow B(x, z) \land C(z, y)\right\}
\end{align*}
\]

\subsection{3.5 Intermezzo: Constants and Free Variables}

The goal of this section is to show how the previous results can be applied to more realistic databases (with constants and nulls) and non-boolean queries (with free variables).

\noindent \textbf{3.5.1 Hard and Soft Constants}

A database $D$ is a set of atoms $R(t_1, \ldots, t_n)$ where each term $t_i$ is either a variable (also known as a labelled null) or a constant from a finite set $\Delta = \Delta_h \cup \Delta_s$, where: $\Delta_h$ is a set of hard constants which are subject to the standard unique name assumption (UNA); and $\Delta_s$ is a set of soft constant which are not subject to the UNA (see e.g. \cite{20}). Given a database $D$ we denote by $D^*$ the instance obtained from $D$ as follows: (1) rename every $c \in \Delta_s$ into a variable $v_c$; (2) for every $c \in \Delta$ introduce a fresh predicate $R_c$ and add the atom $R_c(v_c)$; (3) introduce a fresh predicate $R_o$ and, for all $c, c' \in \Delta_h$ such that $c \neq c'$, add the atom $R_o(v_c, v_{c'})$. This definitions correspond to the standard encoding of constants and it can similarly be applied to boolean queries with constants. The following properties are then readily verified: (1) given a database $D$ and a set $\Sigma$ of tgd and egds, $D \land \Sigma$ is satisfiable iff $D^* \land \Sigma \not\equiv \{R_o(x, x)\}$; (2) when $D \land \Sigma$ is satisfiable, for all set $Q$ of instances, we have $D \land \Sigma \models Q$ iff $D^* \land \Sigma \models Q^*$. A less obvious observation, formalized below, is that this technique of simulation can also be used for query rewriting under integrable egds.

\textbf{Proposition 3.}

\begin{itemize}
\item Given a set of tgd $\Sigma$, an integrable set of functional dependencies $F$, and a database $D$, the formula $D \land \Sigma \land F$ is satisfiable iff $D^* \land \Sigma_F \not\equiv \{R_o(x, x)\}$.
\item Under satisfiability, for all databases $D$, all data rewritings $D_F$ of $(D^*, \Sigma_F)$ and all sets $Q$ of instances, we have $D \land \Sigma \land F \models Q$ iff $D^* \land \Sigma_F \models Q^*$.
\end{itemize}

\textbf{Corollary 2.} Given a set $\Sigma$ of tgd and an integrable set $F$ of functional dependencies, if $\Sigma_F$ is sticky, then, for all set $Q$ of instances, the following problem is in $\text{PTime}$: given a database $D$, does $D \land \Sigma \land F \models Q$?
3.5.2 Free Variables and Equalities

Recall that a query \( Q \) is called a union of conjunctive queries with equalities, denoted \( Q \in \text{UCQ}^m \), when \( Q \) is a first-order query of the form

\[ Q = \{(x_1, \ldots, x_n) | \bigvee_i Q_i \} \]

where each \( x_i \) is a called a free variable and each clause \( Q_i \) is a finite conjunction of relational atoms and equality atoms (with constants, free variables, and existential variables). Given such a query \( Q \), we may consider the set \( Q^* \) of instances obtained as follows: (1) rename every constant \( c \) into a variable \( v \) and add the atom \( R_c(v) \) to each clause; (2) for every free variable \( x \), introduce a predicate \( V_i \) and add the atom \( V_i(x) \) to each clause; (3) introduce a predicate \( R_c \) and replace every equality atom \((t = t')\) by the atom \( R_n(t, t') \); and (4) let \( Q^* \) be the resulting set of clauses (which now consist of instances with variables only). Conversely, given an instances \( R^* \) of the extended schema (and for a fixed tuple \((x_1, \ldots, x_n)\) of free variable) we let \( R \) the set of relational and equality atoms over the original schema which is obtained by replacing every atom \( R_c(u) \) by \((u = c)\) and every atom \( V_i(u) \) by \( u = x_i \). Given a set \( \Sigma^* \) of instances over the extended schema, we finally denote by \( \mathcal{R} \) the UCQ\(^m\) query of the form \( \mathcal{R} = \{(x_1, \ldots, x_n) | \bigvee_i R_i \in \Sigma^* \} \). These definitions are exemplified below:

\[
\Sigma = \{A(u, v) \rightarrow B(u, w)\} \\
Q = \{ (x_1, x_2), (y_1, y_2) \}
\]

\[
Q^* = \{ (B(x_1, x_2), V_i(x_1), V_j(x_2)) \}
\]

\[
R^* = \{R(u, v) \rightarrow \{A(u, v), V_i(u), V_j(u)\}\}
\]

As next formalized, the above technique can be used to generalize the results of the previous sections to the case of non-boolean queries with constants and equalities:

**Proposition 4.** Given \( \Sigma, Q, Q^*, R^*, \mathcal{R} \) as above where

\[
\mathcal{R}^* \in \text{Fix}(Q^*, \Xi_{\text{UD}})
\]

the UCQ\(^m\) query \( \mathcal{R} \) is a rewriting of the UCQ\(^m\) query \( Q \). More formally, for all database \( D \) and all tuples \( \bar{c} \) of constants, the following statements are equivalent:

- \( \bar{c} \) is an answer of \( \mathcal{R} \) in \( D \), denoted \( \bar{c} \in \mathcal{R}(D) \)
- \( \bar{c} \) is a certain answer of \( Q \) in \( D \) under \( \Sigma \), meaning that \( \bar{c} \in Q(D') \) for all instance \( D' \) such that \( D \wedge \Sigma \models D' \).

**Corollary 3.** We can use the technique of saturated resolution to compute a finite UCQ\(^m\) rewriting for a given UCQ\(^m\) query whenever there exists one (e.g. under stickiness).

4. DATALOG REWRITING

As announced in the introduction, we now consider a more general notion of rewriting, called Datalog rewriting, which will prove to be a unifying paradigm of tractability for Datalog\(^\pm\). More precisely, we will show that it captures the class of terminating dependencies (Section 4.1) and the class of guarded tgds (Section 4.2).

**Definition 4.** Given a set \( \Sigma \) of tgds and egds, and a set \( Q \) of instances, a Datalog rewriting for \((\Sigma, Q)\) is a triple \((\sigma_A, \Gamma, G)\) where

- \( \sigma_A \) is a set of predicates which do not occur in \( \Sigma \) or \( Q \);
- \( \Gamma \) is a finite set of tgds \( B \rightarrow H \) where \( \forall H \subseteq \forall B \);
- \( G \) is an instance of the form \( G = \{F()\} \) where \( F \in \Sigma_A \);
- for all \( \sigma_A \)-free instances \( D \), it holds that:

\[
D \wedge \Sigma \models Q \iff D \wedge \Gamma \models G.
\]

Note that, in the above definition, each tgd in \( \Gamma \) corresponds to a standard Datalog rule (also known as a full tgd). Since \( \Gamma \) is required to be finite, \( \Gamma \) corresponds to a standard Datalog program. A predicate of \( \sigma_A \) will be called an auxiliary predicate and \( \sigma_A \) corresponds intuitively to an intentional schema. An \( \sigma_A \)-free instance is defined as an instance in which no predicate of \( \sigma_A \) occurs. That is, an \( \sigma_A \)-free instance corresponds intuitively to an extensional database.

The instance \( G \) is finally known as the goal of the Datalog program \((\sigma_A, \Gamma, G)\) and the predicate \( G \), of arity 0, is known as the goal predicate. The following proposition finally summarizes the basic properties of Datalog rewritings:

**Proposition 5.**

- If a Datalog rewriting exists for \((\Sigma, Q)\), the following problem is in \( \text{Ptime} \): given \( D \), does \( D \wedge \Sigma \models Q \)?
- If there is a finite query rewriting for \((\Sigma, Q)\) then there is also a Datalog rewriting for \((\Sigma, Q)\).
- There are some pairs \((\Sigma, Q)\) for which a Datalog rewriting exists while no finite query rewriting exists.

**Proof Sketch.** The first point follows from the following observation: when \( \Gamma \) is a set of full tgds, we can compute a data rewriting \( \mathcal{U} \) for \((D, \Gamma)\) in polynomial time (for a fixed \( \Gamma \) using the chase, and we can then test in polynomial time (for a fixed instance \( G \)) whether \( \mathcal{U} \models G \). For the second point, given a finite query rewriting \( \mathcal{R} \) for \((\Sigma, Q)\) and with letting \( G = \{G()\} \) for some fresh predicate \( G \), we can observe that \((\{G\}, \{G \rightarrow \bigcup R, G\})\) is a Datalog rewriting for \((\Sigma, Q)\). Finally, for \( \Sigma := \{R(x, y), R(y, z) \rightarrow \bigcup R(x, z)\} \) and \( Q = \{R(x, x)\} \), the pair \((\{G\}, \Sigma \cup \{R(x, x) \rightarrow G\})\) is a Datalog rewriting of \((\Sigma, Q)\) while there is no finite (first-order) query rewriting for \((\Sigma, Q)\).

4.1 From Termination To Datalog

This section revisits the criterion of oblivious termination which was introduced in [20] and presented in [10] as a language of the Datalog\(^\pm\) family. As discussed in [10], there are alternative criteria of termination that can be considered (see [20] for the current state of the art). Note however that oblivious termination captures the case of weakly acyclic [15] sets of tgds and arbitrary sets of egds, and the results presented in this section can be extended to the classes discussed in [20]. In a nutshell, oblivious termination is based on (1) a technique of simulation that encodes the egds by means of tgds and (2) a standard notion of skeletization which generates a set of rules with function symbols (that is, a logic program). As observed in [20], this logic program enjoys a technical bounded depth property. In turn, we will show in this section that this bounded depth property ensures the existence of a Datalog rewriting.

**Simulation.** Given a set \( \Sigma \) of tgds and egds, we say that \( \Sigma' \) is a substitution-free simulation of \( \Sigma \), denoted \( \Sigma' \in \)}
Sim$_E$(Σ), when Σ' is a set of tgds obtained from Σ using the simulation technique from [20] (which, unlike alternative techniques, avoid the use of substitution axioms). More precisely, we let Σ' ∈ Sim$_E$(Σ) when E is a binary predicate which does not occur in Σ and Σ' can be computed with the following non-deterministic algorithm:

Start from Σ' = Σ.
Add the following tgds to Σ':
E(x, y) → E(y, x)
E(x, y), E(y, z) → E(x, z)
For all predicates R occurring in Σ (of arity n)
Add the following tgd to Σ':
R(x_1, ..., x_n) → E(x_1, x_1), ..., E(x_n, x_n)
Repeat (until fixed point)
If there is a tgd B → H or an edg B → x=y in Σ' and a variable x ∈ V_B that occurs more than once in B
Let x' be a fresh variable
Replace one occurrence of x by x' in B
Add the atom E(x, x') to B
For all egds r : B → x = y in Σ'
Replace r by the tgd B → E(x, y)
Return Σ'.

Skolemization. Given a set of tgds Σ, we denote by P$_{Σ}$ the logic program which is obtained by skolemizing Σ is a standard way. That is, for all B(X, Y) → H(X, Z) in Σ, the program P$_{Σ}$ contains a rule B(X, Y) → H'(X) where H' is obtained from H by replacing every variable z ∈ Z by a term f(¯x) where f is function symbol and the tuple ¯x is a fixed ordering of X = {x}. Given an instance I and such a logic program P$_{Σ}$ we then denote by P$_{Σ}$(I) the fixed point of I and P$_{Σ}$ (also known as the minimal Herbrand model).

Definition 5.
- We say that a set Σ of tgds ensures oblivious termination iff, for all finite instance I, P$_{Σ}$(I) is finite.
- Given a set Σ of tgds and egds, we say that Σ ensures oblivious termination iff there exists Σ' ∈ Sim$_E$(Σ) such that Σ' ensures oblivious termination.

Bounded Depth. Given a skolem term t with variables and function symbols, we define the depth d(t) of t in a standard way. More precisely, given a variable x we let d(x) = 1, and given a term t = f(t_1, ..., t_n), we let d(t) = 1 + max$d_i$(d(t_i)). Given a set of tgds Σ, an instance D and an integer k we denote by P$_{Σ}^k$(D) the set of atoms with skolem terms which is obtained by applying (inductively) all the rules B(X, Y) → H'(X) of P$_{Σ}$, but only for the valuations θ of X ∪ Y such that, for all u ∈ X ∪ Y, the depth of θ(u) is at most k. It can be checked that P$_{Σ}^k$(I) is finite and well-defined whenever Σ, I and k are finite. In particular, the order of application of the rules does not matter. Note here that P$_{Σ}^k$(I) is a skolem instance (with skolem terms) rather than a standard instance (with variables only) but he definitions from Section 3.1 can nonetheless be extended in a natural way. In particular, given a set Σ of instances, we let P$_{Σ}^k$(I) = Σ if, for all Q ∈ Σ, there is a mapping θ from V_Q to the terms of P$_{Σ}^k$(I) such that Q[θ] ⊆ P$_{Σ}^k$(I).

Definition 6. Given a set Σ of tgds and a set Σ of instances, we say that (Σ, Q) has bounded depth iff there exists a finite integer k (depending only on (Σ, Q)) such that, for all instances D, the following statements are equivalent:

- D ∩ Σ |= Q
- P$_{Σ}^k$(D) |= Q

The following result was established in [20]:

Lemma 2. If Σ is a finite set of tgds ensuring oblivious termination, there exists k (depending only on Σ) such that, for all instances D, P$_{Σ}^k$(D) = P$_{Σ}^k$(D). Therefore, for all sets Q of instances, (Σ, Q) has bounded depth.

We next present the main result of this section.

Theorem 7. Given a set Σ of tgds and a set Q of instances, if (Σ, Q) has bounded depth (and if we know the bound k), we can compute a Datalog rewriting for (Σ, Q).

Proof Sketch. The key idea of the proof is to use fresh predicate symbols to simulate the effect of the function symbols of P$_{Σ}^k$. Intuitively, every atom R(t) with skolem terms can indeed be simulated with a standard atom R_σ(¯x), with variables only, where σ encodes the “shape” of each term, while ¯x corresponds to the variables that occur in t. For instance:

\[ R(x, f(x, y), g(y, f(x, z))) \approx R_1(f(1,2), g(2,f(1,3)), x, y, z) \]

for a fixed bound k, for every rule B(X, Y) → H'(X) in P$_{Σ}$ and for every valuation θ of X∪Y such that max$\{d(θ(u))|u ∈ X ∪ Y\}$ ≤ k we can then translate every skolem atom in the rule \[ B[θ] → H[θ] \] into a standard atom. For instance, if k = 2 and P$_{Σ}$ contains only one binary skolem function f, we can replace the rule

\[ A(x, y) → B(x, y, f(x, y)) \]

by a set of Datalog rules containing, among others, the rules:

\[ A(x, y) → B_1,2, f(1,2), x, y) \]
\[ A_1, f(1,1), f(x) → B_1, f(1,1), f(1, f(1,1)), x) \]
\[ A_1, f(1,2), f(x, y) → B_1, f(1,2), f(1, f(1,2)), x, y) \]
\[ A_1, f(1,2), f(1,3), f(x, y, z) → B_1, f(1,3), f(1,2, f(1,3)), x, y) \]

... Consider now Q of the form Q = \{G(1)\} such that (Σ, Q) has bounded depth k. Let σ$_A$ be the set predicates R_σ where σ encodes a shape of depth ≤ k. Let Γ$^B$ be set of Datalog rules resulting from the above construction. We can check in this case that (σ$_A$, Σ, {G(1)}) is a Datalog rewriting for (Σ, Q). In the general case, (when Q is not already of the form \{G(1)\}), we may introduce a fresh 0-ary predicate G, consider all the tgds r_Q : Q → G(I) where Q ∈ Σ, and consider all the valuations θ of X_Q such that max$\{d(θ(x))|x ∈ X_Q\}$ ≤ k. We can then encode each of these rules with a Datalog rule over σ$_A$ ∪ \{G\}, and we can conclude as in the previous case. □

Corollary 4. There is an algorithm that, given a set Σ of tgds and egds ensuring oblivious termination, and given a set Q of instances, computes a Datalog rewriting for (Σ, Q).

4.2 From Guardedness To Datalog

In this section, we consider the class of guarded tgds. Intuitively, we will say that a tgd is guarded when there is an atom in the body (called a guard) that contains all the universal variables. In turn, a variable is called universal iff it occurs both in the body and the head. Note that the variables that occur only in the body are not taken into account in this definition of guardedness. Therefore, the class
of guarded tgds contains, as a special case, the tgds $B \rightarrow G()$ where $B$ is an arbitrary instance and $G$ is a 0-ary predicate (for example, a goal predicate) because such tgds have no universal variable. Another example of guarded tgd is

\[ A(x, y, z), B(i, x, y), B(j, y, z) \rightarrow \exists k, B(k, x, z) \]

where the set of universal variables is \{x, z\} and the atom $A(x, y, z)$ is a guard. In contrast, the tgd

\[ B(i, x, y), B(j, y, z) \rightarrow \exists k, B(k, x, z) \]

in not guarded since there is no atom in the body that contains both $x$ and $z$.

**Definition 7.** A tgd of $B \rightarrow H$ is guarded iff there is an atom $G \in B$ such that $(V_B \cap V_H) \subseteq V_G$.

### 4.2.1 The Case of $\beta$-Guardedness

A special class of guarded tgds was considered in [8, 10], called $\beta$-guarded tgds in this section, that complies with the following definition:

**Definition 8.** A tgd $B \rightarrow H$ is called $\beta$-guarded iff there exists an atom $G \in B$ such $V_B \subseteq V_G$.

Note that a $\beta$-guarded tgd is (only) a special case of guarded tgds since the requirement $V_B \subseteq V_G$ is stronger than $(V_B \cap V_H) \subseteq V_C$. For example, the tgd

\[ A(x, y, z), B(i, x, y), B(j, y, z) \rightarrow \exists k, B(k, x, z) \]

is guarded but not $\beta$-guarded. While $\beta$-guardness is slightly less general, it was shown in [8] that the class of $\beta$-guarded tgds remains a very natural class to consider. In particular, it was shown in [8] that we only need $\beta$-guardness (as opposed to general guardedness) to cover interesting classes of ontologies, including languages from the DL-lite family [3]. Another important advantage of $\beta$-guardness is that it ensures a useful property, established in [8], called the bounded guard-depth property. This property (defined for $\beta$-guarded tgds only) proves indeed very relevant here as it coincides in fact with the general property of bounded depth which was discussed in the previous section. Combining the results in [8] with this observation, we obtain the following results:

**Lemma 3.** If $\Sigma$ is a finite set of $\beta$-guarded tgds and $Q$ is a finite set of instance, then $(\Sigma, Q)$ has bounded depth $k$ for some computable $k$ that depends both on $\Sigma$ and $Q$.

**Corollary 5 (of Theorem 7).** There is an algorithm that, given a set $\Sigma$ of $\beta$-guarded tgds and a set $Q$ of instances, computes a Datalog rewriting for $(\Sigma, Q)$.

As already discussed, a $\beta$-guarded tgd is only a special case of guarded tgd, and the work of [8] suggests that there is no trivial reduction from the class of guarded tgds to the class of $\beta$-guarded tgds. This is why we consider an alternative approach, in the following section, for the more general case.

### 4.2.2 From Guardedness To Flatness

In this section, we provide a proof (sketch), based on a technical notion of flatness, for the following result:

**Theorem 8.** For every finite set $\Sigma$ of guarded tgds and set $Q$ of instances, there is a Datalog rewriting for $(\Sigma, Q)$.

The key idea of the proof can be summarized as follows: when $\Sigma$ is a set of guarded tgds, there exists an equivalent set of tgds $\Sigma'$ which enjoys the flat chase property. This property meaning intuitively that it is sufficient to chase the tgds $B(X, Y) \rightarrow H(X, Z)$ of $\Sigma'$ for the renamings of $X$ that map $X$ to the variables of the original instance $D$ (which intuitively correspond to constant values). In turn, the flat chase property can be linked with the property of bounded depth (for the depth $k = 1$) and Theorem 7 can be used again to prove the existence of a Datalog rewriting.

**Flat Chase.** Consider a triple $(D, I, J)$ of instances and a set $\Sigma$ of tgds. When $I \overset{\text{flat}}{\Rightarrow}_\Sigma J$ we say that the chase step is flat with respect to $D$, denoted $I \overset{\text{flat}}{\Rightarrow}_D J$, when there is a tgd $B(X, Y) \rightarrow H(X, Z)$ in $\Sigma$ and two substitutions $\theta_X$ and $\theta_Y$ such that:

- $B[\theta_X, \theta_Y] \subseteq I$ and $J = I \cup H[\theta_X, \theta_Y]$ for some $I$-fresh renaming $\theta_Z$; and
- in addition, for all $x \in X$, it holds that $\theta_X(x) \in V_B$.

Given an instance $U$, we finally let $U \in \text{Flat}(D, \Sigma)$ when there exists a finite derivation of the form

\[ D = I_1 \overset{D}{\Rightarrow}_\Sigma I_2 \overset{D}{\Rightarrow}_\Sigma \ldots \overset{D}{\Rightarrow}_\Sigma I_n = U. \]

**Definition 9.** Given a set $\Sigma$ of tgds and a set $Q$ of instances, we say that $(\Sigma, Q)$ has the flat chase property iff, for all instance $D$ the following statements are equivalent:

- $D \land \Sigma \models Q$
- $\exists U \in \text{Flat}(D, \Sigma), U \models Q$

**Lemma 4.** If $(\Sigma, Q)$ has the flat chase property, then $(\Sigma, Q)$ has bounded depth (for the depth $k = 1$) and therefore, there exists a Datalog rewriting for $(\Sigma, Q)$.

The proof of Theorem 8 finally relies on Lemma 5 below.

**Lemma 5.** For all finite set $\Sigma$ of guarded tgds and all finite set $Q$ of instances, there exists a finite set $\Sigma'$ of tgds such that $\Sigma \equiv \Sigma'$ and $(\Sigma', Q)$ has the flat chase property.

**Proof Sketch.** Given a tgd $r = B(X, Y) \rightarrow H(X, Z)$, a refinement of $r$ is a tgd $r' = B' \rightarrow H'$ where $B' = B[\theta_X, \theta_Y]$ and $H[\theta_X, \theta_Y]$ for some renamings $\theta_X, \theta_Y$ where $\theta_Z$ is a $B'$-fresh renaming of $Z$. We say that $r'$ is a careful refinement of $r$ when, in addition, $\theta_Y$ is a $H'$-fresh renaming.

Given a set of tgds $\Sigma$, we let $\text{Ref}(\Sigma)$ (resp. $\text{CRef}(\Sigma)$) the sets of tgds corresponding to a refinement (resp. careful refinement) of some tgd of $\Sigma$. Given a tgd $r : B \rightarrow H$ we say that $r$ is of the split form

\[ r : (G(X, U), B'(X', U', V) \rightarrow H(X, Z) \]

when $X$ and $Z$ are defined as usual, $G$ is a subset of $B$ satisfying $X \subseteq V_G, U$ is the set of remaining variables in $G$, $B'$ is the set of atoms in $B'(G, V)$ is the set of variables that occur only in $B'$, and finally $(X', Y') = ((X \cap V_B)' \cup (Y \cap V_B))$.

Given two sets $\Sigma_1$ and $\Sigma_2$ of tgds, we say that a tgd $r_2$ is derived from $(\Sigma_1, \Sigma_2)$ when there exists a tgd $r_1 \in \text{CRef}(\Sigma_1)$ and a tgd $r_2 \in \text{Ref}(\Sigma_2)$ such that:

- $r_1 : B_1(X_1, Y_1) \rightarrow H_1(X_1, Z_1);$
Given a set of tgds $\Sigma$, we define the flattening of $\Sigma$, denoted $\text{flat}(\Sigma)$, the body of $i$ $k$

$X$ $3$

Proof Sketch. Fact 2. When a tgd $r$ is derived from $(\Sigma_1, \Sigma_2)$, it holds that $\Sigma_1 \cup \Sigma_2 \vdash r_3$

Fact 1. When a tgd $r_3$ is derived from $(\Sigma_1, \Sigma_2)$, it holds that $\Sigma_1 \cup \Sigma_2 \vdash r_3$

Proof Sketch. Using the previous notations, we have

$(B_1 \cup B_2') \cap \text{chase}_{\text{ref}}(r_1)(B_1 \cup B_2' \cup H_1) \cap \text{chase}_{\text{ref}}(r_2)(B_1 \cup B_2' \cup H_1 \cup H_2)$

and it follows that $\{r_1, r_2\} \vdash r_3$.

Given an atom $k$ we say that a tgd $r$ is $k$-guarded if $r$ is of the form

$r : G(X, U), B'(X', U', V) \rightarrow H(X, Z)$

for some instance $G$ with only one atom (that is, a guard atom), and there exists a set of instances $(B')_{i=0}^{n}$ called a $(G, k)$-decomposition of $B$ such that: (1) $B' = \cup_{i<n}(B')$, (2) for each $i \leq n$, $B'$ has at most $(k - 1)$ atoms; and (3) for all $i \neq j$, it holds that $V_{B'_{i}} \cap V_{B'_{j}} \subseteq V_{C_{i}}$. We define the guard width of a guarded tgd $r$, denoted $\text{gw}(r)$ as the smallest $k$ such that $r$ is $k$-guarded. In contrast, we define the left width of a tgd $r$, denoted $\text{lw}(r)$, as the number of atom in the body of $r$. Note that, every guarded tgd is such that $\text{gw}(r) \leq \text{lw}(r)$. Given a set of tgds $\Sigma$, we finally let $\text{gw}(\Sigma) = \max\{\text{gw}(r), r \in \Sigma\}$ and $\text{lw}(\Sigma) = \max\{\text{lw}(r), r \in \Sigma\}$.

Fact 2. When a tgd $r_3$ is derived from $(\Sigma_1, \Sigma_2)$, it holds that $\text{gw}(r_3) \leq \text{max}(\text{gw}(\Sigma_1), \text{lw}(\Sigma_2))$.

Proof Sketch. It can be checked that $\text{lw}(r') \leq \text{lw}(r)$ whenever $r'$ is a refinement of $r$ and $\text{gw}(r') \leq \text{gw}(r)$ whenever $r'$ is a careful refinement of $r$. Let $r_1 \in \text{CRef}(\Sigma_1)$ be a $k$-guarded tgd of form

$r_1 : G_1(X_1, U_1), B'_1(X'_1, U'_1, V'_1) \rightarrow H_1(X_1, Z_1)$

and a $(G_1, k)$-decomposition $B'_1 = \cup_{i<k}B'_1$. Let $r_2 \in \text{Ref}(\Sigma_2)$ such that $\text{lw}(r_2) \leq k$ and $r_2$ is of the form

$r_2 : G_2(X_2, U_2), B'_2(X'_2, U'_2, V'_2) \rightarrow H_2(X_2, Z_2)$.

Finally, let $r_3$ be the tgd derived from $r_1$ and $r_2$, and recall that $r_3$ is of the form

$r_3 : G_3(X_3, U_3), B'_3(X'_3, \theta, V_3) \rightarrow H_3(X_3, Z_3)$

where $G_3 = G_1 \cup B'_1, X_3 = X_1, U_3 = U_1 \cup V_1$, and $B'_3 = B'_2$. Since $\text{lw}(r_2) \leq k$ and $G_2$ is non-empty, we have $|B'_2| \leq (k - 1)$. For all $i \leq n$, we have $V_{B'_i} \cap V_{B'_j} \subseteq X_i \subseteq V_{C'_i}$.

Therefore, the set of instances $\{B'_1\}_{i\leq k} \cup \{B'_2\}$ is a $(G, k)$-decomposition of the body of $r_3$ and $r_3$ is $k$-guarded.

Given a set of tgds $\Sigma$, we define the flattening of $\Sigma$, denoted $\Sigma'$, as the minimal set of tgds $\Sigma'$ such that $\Sigma \subseteq \Sigma'$ and $\Sigma'$ contains all the tgds that can be derived from $\Sigma$.

Fact 3. For all set $\Sigma$ of tgds and all set $Q$ of instances, $(\Sigma^\infty, Q)$ has the flat chase property.

Proof Sketch. Consider an instance $D$ and suppose that $D \cap \Sigma \models Q$. By completeness of the chase, we know that there exists a derivation

$D = I_0 \quad \text{chase}_{\Sigma_1} \quad I_1 \quad \text{chase}_{\Sigma_2} \quad \ldots \quad \text{chase}_{\Sigma_n} I_n$

such that $I_n \models Q$. If this derivation is flat, and since $\Sigma \subseteq \Sigma^\infty$, we have $I_n \in \text{Flat}(D, \Sigma^\infty)$ and $I_n \models Q$. Suppose now that the derivation is not flat and consider the first integer $j \leq n$ such that $I_j$ is obtained from $I_{j-1}$ with a chase step which is not flat. We can see that there is a tgd $r_2 : B_2(X_2, U_2) \rightarrow H_2(X_2, Z_2)$ in $\text{Ref}(\Sigma_2)$ such that $B_2 \subseteq I_{j-1}$, $I_j = I_{j-1} \cup H_2$ and $X_2 \nsubseteq V_D$. Consider a guard atom $G(\theta)$ for $r_2$. Since $X \nsubseteq \{\theta\}$ and $X \nsubseteq V(D)$, there exits some $i \leq j$ such that $G(\theta) \in \{I_i \setminus I_{i-1}\}$. Let $G_2$ be the set of all the atoms of $B_2$ that belong to $(I_j \setminus I_{j-1})$. Write $r_2$ under the form

$r_2 : G_2(X_2, U_2), B'_2(X'_2, U'_2, V'_2) \rightarrow H_2(X_2, Z_2)$.

and observe that $G_2$ is non-empty (since it contains $G(\theta)$). Considering $N_j = V_{\theta} \setminus V_{I_{j-1}}$, since the derivation is flat from the step 1 to the step $j$, we can observe that the only atom of $I_{j-1}$ containing a variable of $N_j$ are the atoms of $I_j \setminus I_{j-1}$. Therefore, $(U'_2 \cup V_2)$ is disjoint from $N_j$. By definition of careful refinements, we can now consider a tgd $r_1 \in \text{CRef}(\Sigma)$ of the form

$r_1 : B_1(X_1, Y_1) \rightarrow H_1(X_1, Z_1)$

where $Y_1$ is set of variables disjoint from $(X'_2, U'_2, V_2), B_1[\theta_{Y_1}] \subseteq I_{j-1}$ for some renaming $\theta_{Y_1}$ of $Y_1$, and $I_1 = I_{j-1} \cup H_1$. Let $r_3 = B_3 \rightarrow H_3$ be the tgd such that $B_3 = B_1 \cup B'_2$ and $H_3 = H_1 \cup H_2$, and observe that $r_3 \in \Sigma^\infty$. Let $\theta'$ be an injective renaming of $Z_1 \cup Z_2$ into a set of fresh variables (disjoint from $V_{I_{j-1}}$) and for all $k \leq j$, let $I'_k = I_k[\theta']$. We can observe that we have

$D = I_0 \text{chase}_{\Sigma_1} \ldots \text{chase}_{\Sigma_n} I_{j-1} \text{chase}_{\Sigma_3} I'_3 \text{chase}_{\Sigma_3} \ldots \text{chase}_{\Sigma_n} I_n$

where the step $I_{j-1} \text{chase}_{\Sigma_3} I'_3$ is now a flat step.

We can finally generalize the above construction to show (by induction on $n$) that every chase derivation of $D$ by $\Sigma$ (leading to $I_n \models Q$) can be transformed into a flat derivation of $D$ by $\Sigma^\infty$ (leading to $I_n \models Q$).

Given a tgd $r : B(X, Y) \rightarrow H(X, Z)$, we define a left-core projection of $r$ as a minimal (for $\subseteq$) set of atoms $B' \subseteq B$ such that $B' = B[\theta_Y]$ for some renaming $\theta_Y$ of $Y$. Given a set of tgds $\Sigma$, we let $\text{LC}(\Sigma)$ be the set of instances $B'$ such that $B'$ is a left-core projection of some tgd $r \in \Sigma$.

Fact 4. Given a finite schema $\sigma_0$, an integer $k_0$, and infinite set $\Sigma$ of $k_0$-guarded tgds over $\sigma_0$, the set $\text{LC}(\Sigma)$ is finite up to isomorphism.

Proof Sketch. On a fixed schema $\sigma_0$, there are (up to isomorphism) only a finite number of atoms that can be used as a guard $G$ for a decomposition $(B')_{i<k_0}$. For a fixed integer $k_0$, and for every instance $B = G \cup \{B'\}$ corresponding to the left-core projection of some tgd, there is only a finite number of possible blocks $B'$ satisfying $|B'| \leq k - 1$ that can be used in the composition, up to bijective renaming of $V_{\theta} \setminus G$. It follows that each $B \in \text{LC}(\Sigma)$ is finite and that $\text{LC}(\Sigma)$ is finite up to isomorphism.
Given a set of tgd $\Sigma$ and an integer $k$, we that that a tgd $r : B \rightarrow H$ is a $k$-tgd of $\Sigma$, denoted $\Sigma^{(k)}$ when there exists a tgd $B' \rightarrow H'$ in the flating $\Sigma^{\infty}$ of $\Sigma$ such that: (1) $H$ is a subset of $H'$ containing at most $k$ atoms; and (2) $B$ is a left-core projection of $B' \rightarrow H$. As a corollary of Fact 4, it can be observed that $\Sigma^{(k)}$ is finite (or isomorphism) whenever $\Sigma$ is a finite set of guarded tgds.

**Fact 5.** For all sets $\Sigma$ of guarded tgds, all sets $Q$ of instances and all integer $k$ such that

$$k \geq \max\{|B|, B \in LC(\Sigma^{\infty}) \cup Q\}$$

$(\Sigma^{(k)}, Q)$ enjoys the flat chase property.

**Proof Sketch.** The property can be shown by adapting the proof of Fact 3. Indeed, a flat derivation by $\Sigma^{\infty}$ can be modified in two ways that preserves completeness: (1) one may replace the body of a tgd by a left-core of this tgd; and (2) one may select, for each tgd $B \rightarrow H$, the atoms from $H$ which will be used later in the derivation (as to enforce that $|H| \leq k$).

We can finally use Fact 5 to conclude the proof of Lemma 5 and Theorem 8.

## 5. CONCLUSION AND FUTURE WORK

This paper presented contributions along two axes:

**More General Classes.** Considering a complete resolution procedure extends the possible fields of applications of Datalog±. Resolution can be used, first, to generalize existing tractability criteria (such as stickiness) but it can also be used, in practice, without any syntactic assumption. A technique of integration or simulation however proves often useful (and sometimes necessary) to handle egds. Datalog rewriting finally covers the three main paradigms of Datalog±: stickiness, termination and guardedness. There are also alternative paradigms of tractability that were considered in [2] which have not been discussed here (to simplify the discussion) but are yet to compare with the class of Datalog-rewritable dependencies. In particular, a natural question (left as future work) is the following: is Datalog rewriting always possible under bounded-tree width?

**More Efficient Algorithms.** This paper introduced several algorithms and heuristics that can be of clear practical use. In particular, the resolution can certainly (at least, in some contexts) be more efficient than the chase. The technique of Datalog Rewriting can also prove useful in practice. In particular, one may consider Magic Set or any other standard optimisation of Datalog to improve efficiency. Nonetheless, there is still a long road ahead because the algorithms of Datalog Rewriting proposed in Section 4 are (for the moment) very much non-optimal. An interesting and challenging question that remains is the following: how to compute in practice a Datalog rewriting of reasonable size?

## 6. REFERENCES

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

[2] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *J. Comput. Syst. Sci.*, 43(1):62–124, 1991.

[3] A. Acciari, D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QuOnto: Querying ontologies. In *AAAI*, pages 1670–1671, 2005.

[4] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. Extending decidable cases for rules with existential variables. In *IJCAI*, pages 677–682, 2009.

[5] F. Barany, G. Gottlob, and M. Otto. Querying the guarded fragment. In *LICS*, 2010.

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

[7] F. Bry, T. Furch, B. Marnette, C. Ley, B. Linse, and O. Poppe. SPARQL: SPARQL with rules and quantification. In *Semantic Web Information Management: A Model-Based Perspective*, pages 341–369. Springer, 2010.

[8] A. Cali, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *KR*, pages 70–80, 2008.

[9] A. Cali, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. In *PODS*, pages 77–86, 2009.

[10] A. Cali, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris. Datalog+-/: A family of logical knowledge representation and query languages for new applications. In *LICS*, pages 228–242, 2010.

[11] A. Cali, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in datalog+-/. In *RR*, pages 1–17, 2010.

[12] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.

[13] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[14] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

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

[16] E. Gradel. Decision procedures for guarded logics. In *CADE*, pages 31–51, 1999.

[17] J. Hladik. Implementation and optimisation of a tableau algorithm for the guarded fragment. In *TABLEAUX*, pages 145–159, 2002.

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

[19] Y. Kazakov and H. de Nivelle. A resolution decision procedure for the guarded fragment with transitive guards. In *IJCAR*, volume 3097 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2004.

[20] B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.

[21] B. Marnette. *Tractable Schema Mappings Under Oblivious Termination*. PhD thesis, Oxford University, 2010.

[22] B. Marnette, G. Mecca, and P. Papotti. Scalable data exchange with functional dependencies. *PVLDB*, 3(1):105–116, 2010.

[23] A. Nash, P. A. Bernstein, and S. Melnik. Composition of mappings given by embedded dependencies. In *PODS*, pages 172–183, 2005.

[24] H. D. Nivelle and M. D. Rijke. Deciding the guarded fragments by resolution. *Journal of Symbolic Computation*, 35:21–58, 2003.

[25] H. Pérez-Urbina, B. Motik, and I. Horrocks. Rewriting Conjunctive Queries over Description Logic Knowledge Bases. In *Proc. of the Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008)*, Nantes, France, March 2008. Springer.

[26] F. Spezzano and S. Greco. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.