TLP **13** (4–5): Online Supplement, July 2013.
URL: http://dx.doi.org/10.1017/S1471068413000550

Combining decidability paradigms for existential rules

GEORG GOTTLOB
Department of Computer Science, University of Oxford, UK
(*e-mail:* georg.gottlob@cs.ox.ac.uk)

MARCO MANNA
Department of Mathematics and Informatics, University of Calabria, Italy
(*e-mail:* manna@mat.unical.it)

ANDREAS PIERIS
Department of Computer Science, University of Oxford, UK

submitted 10 April 2013; accepted 23 May 2013

Abstract

Existential rules are Datalog rules extended with existential quantifiers in rule-heads. Three fundamental restriction paradigms that have been studied for ensuring decidability of query answering under existential rules are weak-acyclicity, guardedness and stickiness. Towards the identification of even more expressive decidable languages, several attempts have been conducted to consolidate weak-acyclicity with the other two paradigms. However, it is not clear how guardedness and stickiness can be merged; this is the subject of this paper. A powerful and flexible condition, called tameness, is proposed, which allows us to consolidate in an elegant and uniform way guardedness with stickiness.

KEYWORDS: datalog extensions, query answering, decidability, complexity, guardedness, stickiness

1 Introduction

The interest in using logic in databases gave rise to the field of deductive databases. It appeared that logic programming (LP) was a suitable formalism for querying relational databases. In this context, the LP-based query language Datalog has been defined and intensively studied; see, e.g., (Ceri *et al.* 1990). Interestingly, Datalog gone beyond its original purpose, and is now used in a variety of applications including web data extraction (Gottlob and Koch 2004), source code querying and program analysis (Hajiyev *et al.* 2006), and distributed system analysis (Marczak *et al.* 2012). Furthermore, since Datalog rules are clauses in the function-free Horn fragment of first-order logic, Datalog revealed itself relevant also for semantic web applications such as ontological modeling and reasoning. As a consequence, Datalog has evolved

into a first class formalism with efficient implementations such as cmodels (Lierler and Maratea 2004), DLV (Leone *et al.* 2006) and clasp (Gebser *et al.* 2007).

Although Datalog is a powerful rule-based formalism, one of its main weaknesses, already criticised by Patel-Schneider and Horrocks (2007), is its inability to infer the existence of new objects which are not already in the extensional database. *Existential rules*, also known as *tuple-generating dependencies (TGDs)* and *Datalog*$^{\pm}$ *rules*, overcome this limitation by extending Datalog with existential quantification in rule-heads (Baget *et al.* 2011; Krötzsch and Rudolph 2011; Calì *et al.* 2012; Calì *et al.* 2012; Leone *et al.* 2012). Notice that, in the context of the present paper, a set of existential rules can be seen as a logic program since each existentially quantified variable in the head of a rule can be appropriately replaced by a functional term; more details are given in Section 2. Unfortunately, without syntactic restrictions, the above extension leads to undecidability (Beeri and Vardi 1981; Calì *et al.* 2008).

In this context, a query is not just answered against an extensional database D, as in the classical setting, but against a logical theory constituted by D and a set Σ of existential rules. Thus, for a Boolean conjunctive query (CQ) q, one checks whether the logical theory $D \cup \Sigma$ entails q, rather than just checking whether D entails q. Analogously, if q is a CQ $p(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$ with output variables $\mathbf{X}$, then its answer against $D \cup \Sigma$ consists of all tuples $\mathbf{t}$ of constants such that, when we substitute the variables $\mathbf{X}$ with $\mathbf{t}$, $\varphi(\mathbf{t}, \mathbf{Y})$ evaluates to true in every (possibly infinite) model of $D \cup \Sigma$. Answering a CQ q against $D \cup \Sigma$ is equivalent to evaluating the same query over a *universal model* of $D \cup \Sigma$, that is, a model that can be homomorphically embedded into every other model of $D \cup \Sigma$. Such a universal model can be constructed via the well-known *chase algorithm* (Fagin *et al.* 2005; Deutsch *et al.* 2008), which we will present in Section 2. Informally, the chase adds new atoms to the extensional database D, possibly involving null values (Skolem constants), until the final result satisfies Σ. For example, consider the database $D = \{person(john)\}$ and the set Σ consisting of the rules $person(P) \rightarrow \exists F father(F, P)$ and $father(F, P) \rightarrow person(F)$, asserting that every person has a father, and every father is a person. The chase-expansion of D w.r.t. Σ is the infinite set of atoms $\{person(john), father(z_1, john)\} \cup \bigcup_{i=1}^{\infty} \{person(z_i), father(z_{i+1}, z_i)\}$, where $z_1, z_2, \ldots$ are (labeled) nulls representing unknown individuals. The Boolean conjunctive query $q : p \leftarrow father(X, john), person(X)$, which asks whether John's father is a person, is positively answered on this infinite expansion, and indeed $D \cup \Sigma$ entails q; however, D does not entail q.

The discovery of expressive decidable fragments of TGDs is currently a field of intense research in the AI and KR communities. Several abstract (a.k.a. semantic) classes have been studied so far: *finite expansions sets* (fes), i.e., sets of TGDs which ensure the termination of the chase, *bounded treewidth sets* (bts), i.e., sets which guarantee that the (possibly infinite) instance constructed by the chase has bounded treewidth, and *finite unification sets* (fus), i.e., sets which guarantee the termination of (resolution-based) backward chaining procedures; see (Baget *et al.* 2011). Only recently, *parsimonious sets* (ps), i.e., sets of TGDs under which the chase can be precociously terminated, were introduced (Leone *et al.* 2012). Each one of the above conditions has also its syntactic counterpart: *weakly-acyclic* rules are fes (Fagin *et al.*

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

2005), *guarded-based* rules are bts (Calì *et al.* 2008; Baget *et al.* 2011; Calì *et al.* 2012), *sticky* rules are fus (Calì *et al.* 2012), while *shy* rules are ps (Leone *et al.* 2012). Towards the identification of even more expressive languages, several attempts have been conducted to consolidate the aforementioned classes. Notable formalisms are *glut-guardedness* (Krötzsch and Rudolph 2011) and *weak-stickiness* (Calì *et al.* 2012), obtained by joining weak-acyclicity with guardedness and stickiness, respectively. Unfortunately, none of the above is expressive enough to model real-life cases such as the example below.

Example 1.1
Consider the following set Σ of TGDs:

$\sigma_1 : foEmp(X) \rightarrow \exists Y\ hasMgr(X,Y), foEmp(Y)$
$\sigma_2 : boEmp(X) \rightarrow \exists Y\ hasMgr(X,Y), boEmp(Y)$
$\sigma_3 : hasMgr(X,Y), foEmp(X) \rightarrow foEmp(Y)$
$\sigma_4 : hasMgr(X,Y), boEmp(X) \rightarrow boEmp(Y)$
$\sigma_5 : foEmp(X), foEmp(Y) \rightarrow collOf(X,Y)$
$\sigma_6 : boEmp(X), boEmp(Y) \rightarrow collOf(X,Y)$
$\sigma_7 : ceo(X) \rightarrow foEmp(X), boEmp(X)$
$\sigma_8 : moreSen(X,Y), collOf(X,Y) \rightarrow moreThan(X,Y).$

They, respectively, express that: each front (resp., back) office employee has a manager, who is also a front (resp., back) office employee; the manager of a front (resp., back) office employee is a front (resp., back) office employee; front (or back) office employees are colleagues; the chief executive officer presides both the front and the back office; more senior employees earn more money. The set Σ is neither fes (due to σ_1; on a database as simple as $\{foEmp(a)\}$, the chase does not terminate), nor bts (due to σ_1 and σ_5; the relation *collOf* stores an infinite clique, and thus the instance constructed by the chase has infinite treewidth), nor fus/ps (due to σ_1 and σ_3). ∎

Our goal is to propose new expressive fragments of TGDs that can cope with such real-life scenarios. Let us say that $\Sigma_s = \{\sigma_5, \sigma_6\}$ is sticky, while $\Sigma \setminus \Sigma_s$ is guarded. Both guardedness and stickiness, which we will discuss in Section 2, are well-accepted paradigms. On the one hand, guarded TGDs, inspired by the guarded fragment of first-order logic (Andréka *et al.* 1998), form a robust language which captures important lightweight DLs such as DL-Lite and $\mathcal{EL}$ (Calì *et al.* 2012); in fact, a TGD is guarded if it has a body-atom, called guard, which contains all the body-variables. On the other hand, sticky TGDs allow for joins in rule-bodies which are expressible only via non-guarded rules, and they are able to capture well-known data modeling constructs such as multivalued dependencies (Calì *et al.* 2012). The main research challenge underlying our work is to combine guardedness and stickiness. This is a non-trivial task since we have to join two paradigms which are completely different in nature. Although the techniques in Krötzsch and Rudolph (2011); Calì *et al.* (2012) allow for a natural consolidation of weak-acyclicity with other languages, it is not clear how to merge non-weakly-acyclic formalisms that admit infinite universal models. Hence, we had to come up with novel techniques beyond the state of the art. As a central new paradigm we introduce *tameness*. The key idea underlying this new notion is to tame the interaction between guarded and sticky rules as follows: none of the sticky rules "feeds", during the construction of the chase, the guard atom of a guarded rule; however, sticky rules may "feed" the non-guard atoms. Observe that σ_5 in Example 1.1 may "feed" the atom $collOf(X,Y)$

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

of σ_8; however, by choosing $moreSen(X, Y)$ as the guard atom of σ_8, the tameness condition is satisfied. Our contributions can be summarized as follows:

1. We first investigate the union of guardedness and stickiness. It turns out that query answering under the resulting formalism, called guarded|sticky, is undecidable. In fact, this undecidability result holds even when further restrictions are imposed.
2. We then suggest a natural restriction which is sufficient in order to tame the interaction between guardedness and stickiness, and gain decidability of query answering. Intuitively, we force that none of the sticky rules "feeds" the guard-atom of a guarded rule; however, sticky rules may "feed" the non-guard atoms in an unrestricted way. The above (abstract) condition, which heavily depends on the extensional database, gives rise to a formalism called *tame* guarded|sticky. A sufficient (syntactic) condition, called *predicate-tameness*, that can be checked in polynomial time, is also proposed.
3. We investigate the complexity of query answering under tame guarded|sticky, and we show that the consolidation of guardedness and stickiness comes without paying a price in complexity. In fact, tameness has the same complexity as guardedness, i.e., PTIME-complete in data complexity (only the database is part of the input), NP-complete in case of a fixed set of TGDs, EXPTIME-complete in case of bounded arity, and 2EXPTIME-complete in combined complexity (apart from the database, also the query and the TGDs are part of the input). These results are obtained by providing a novel alternating algorithm.

2 Definitions and background

Technical Definitions. We consider the following pairwise disjoint (infinite) sets: a set Γ of *constants*, a set Γ_N of *labeled nulls*, and a set Γ_V of regular *variables*. We denote by $\mathbf{X}$ sequences (or sets) of variables $X_1, \ldots, X_k$. A *relational schema* $\mathscr{R}$ is a set of *relational symbols* (or *predicates*). A *term* t is a constant, null, or variable. An *atom* has the form $r(t_1, \ldots, t_n)$, where r is a relation, and $t_1, \ldots, t_n$ are terms. For an atom $\underline{a}$, we denote $terms(\underline{a})$, $var(\underline{a})$ and $pred(\underline{a})$ the set of its terms, the set of its variables, and its predicate, respectively; these extend to sets of atoms. Conjunctions of atoms are often identified with the sets of their atoms. An *instance* I for a schema $\mathscr{R}$ is a (possibly infinite) set of atoms $r(\mathbf{t})$, where $r \in \mathscr{R}$ and $\mathbf{t}$ is a tuple of constants and nulls. A *database* D is a finite instance such that $terms(D) \subset \Gamma$. Two sets of atoms A, A' are *S-isomorphic*, where S is a set of terms, denoted $A \simeq_S A'$, if there exists a bijective homomorphism h such that $h(A) = A'$, h^{-1} is a homomorphism, $h^{-1}(A') = A$, and h, h^{-1} are the identity on S.

A *conjunctive query (CQ)* q of arity n over a schema $\mathscr{R}$, written q/n, is an assertion of the form $p(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$, where $\mathbf{X} \cup \mathbf{Y} \subset \Gamma_V$, φ is a conjunction of atoms (possibly with constants) over $\mathscr{R}$, and $p \notin \mathscr{R}$ is an n-ary predicate. Formula φ is the *body* of q, denoted $body(q)$. A *Boolean conjunctive query (BCQ)* is a CQ of arity zero. The *answer* to a CQ q/n over an instance I, denoted $q(I)$, is the set of all n-tuples $\mathbf{t} \in \Gamma^n$

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

for which there exists a homomorphism h such that $h(\varphi(\mathbf{X},\mathbf{Y})) \subseteq I$ and $h(\mathbf{X}) = \mathbf{t}$. A BCQ has a *positive* answer over I, denoted $I \models q$, if $\langle\rangle \in q(I)$.

A *tuple-generating dependency (TGD)* σ over a schema $\mathscr{R}$ is a formula $\forall\mathbf{X}\forall\mathbf{Y}\,\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists\mathbf{Z}\,\psi(\mathbf{X},\mathbf{Z})$, where $\mathbf{X}\cup\mathbf{Y}\cup\mathbf{Z} \subset \Gamma_V$, and φ,ψ are conjunctions of atoms over $\mathscr{R}$; φ is the *body* of σ, denoted $body(\sigma)$, while ψ is the *head* of σ, denoted $head(\sigma)$. For brevity, we will omit the universal quantifiers. An instance I satisfies σ, written $I \models \sigma$, if the following holds: whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X},\mathbf{Y})) \subseteq I$, then there exists $h' \supseteq h$ such that $h'(\psi(\mathbf{X},\mathbf{Z})) \subseteq I$; I satisfies a set Σ of TGDs, denoted $I \models \Sigma$, if I satisfies each $\sigma \in \Sigma$. The *models* of a database D and a set Σ of TGDs, denoted $mods(D,\Sigma)$, is the set of instances $\{I \mid I \supseteq D \text{ and } I \models \Sigma\}$. The *answer* to a CQ q w.r.t. D and Σ, denoted $ans(q,D,\Sigma)$, is the set of tuples $\bigcap_{I\in mods(D,\Sigma)}\{\mathbf{t} \mid \mathbf{t} \in q(I)\}$. The answer to a BCQ q w.r.t. D and Σ is *positive*, denoted $D\cup\Sigma \models q$, if $\langle\rangle \in ans(q,D,\Sigma)$. The problem, called *CQ answering*, tackled in this work is as follows: given a CQ q, a database D, a set Σ of TGDs, and a tuple of constants $\mathbf{t}$, decide whether $\mathbf{t} \in ans(q,D,\Sigma)$. In case that q is a BCQ, the above problem is called *BCQ answering*. The *data complexity* of the above problems is calculated taking only the database as input. The *combined complexity* is calculated considering as input also the query and the set of TGDs. The above problems are LOGSPACE-equivalent (implicit in Chandra and Merlin (1977)), and we focus only on BCQs.

We are going to employ the *chase procedure* (Maier *et al.* 1979; Johnson and Klug 1984), which works on an instance through the TGD *chase rule* defined as follows. Consider an instance I, and a TGD $\sigma : \varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists\mathbf{Z}\,\psi(\mathbf{X},\mathbf{Z})$. We say that σ is *applicable* to I if there exists a homomorphism h such that $h(\varphi(\mathbf{X},\mathbf{Y})) \subseteq I$. Let I' be the instance $I \cup h'(\psi(\mathbf{X},\mathbf{Z}))$, where $h' \supseteq h$ is such that $h'(Z)$ is a "fresh" null not occurring in I, for each $Z \in \mathbf{Z}$. We say that the result of applying σ to I with h is I', and write $I\langle\sigma,h\rangle I'$; in fact, $I\langle\sigma,h\rangle I'$ defines a single TGD *chase step*. The chase algorithm for a database D and a set Σ of TGDs consists of an exhaustive application of TGD chase steps in a fair fashion, which leads to a (possibly infinite) model of D and Σ, denoted $chase(D,\Sigma)$; for the formal definition see the online appendix. In fact, the result of the chase is defined as the least fixpoint of a monotonic operator, that is, the TGD chase step (similar to the immediate consequence operator in logic programming). We denote by $chase^{[k]}(D,\Sigma)$ the instance constructed after $k \geqslant 0$ TGD chase steps. The instance $chase(D,\Sigma)$ is a *universal model* of D and Σ, i.e., for each $I \in mods(D,\Sigma)$, there exists a homomorphism that maps $chase(D,\Sigma)$ to I, and thus $D\cup\Sigma \models q$ iff $chase(D,\Sigma) \models q$, for each BCQ q (Fagin *et al.* 2005). A useful notion, that we are going to employ in our later technical definitions and proofs, is the so-called *chase relation* (Calì *et al.* 2012) of an instance I and a set Σ of TGDs. Roughly, it is a binary relation on atoms, denoted by $CR[I,\Sigma]$, which mimics all the chase derivations of the chase and coincides with the maximum subset of $chase(I,\Sigma)\times chase(I,\Sigma)$ such that $\langle\underline{a},\underline{b}\rangle \in CR[I,\Sigma]$ if $\underline{b}$ is obtained from $\underline{a}$ via a chase step. More formally, assuming that $chase^{[k]}(I,\Sigma)\langle\sigma,h\rangle chase^{[k+1]}(I,\Sigma)$, where $k \geqslant 0$, is applied during the construction of $chase(I,\Sigma)$, and $P_k = h(body(\sigma)) \times (chase^{[k+1]}(I,\Sigma) \setminus chase^{[k]}(I,\Sigma))$, the chase relation $CR[I,\Sigma]$ of I and Σ is the set $\bigcup_{i>0} P_i$. The transitive closure of $CR[I,\Sigma]$ is denoted by $CR^+[I,\Sigma]$.

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

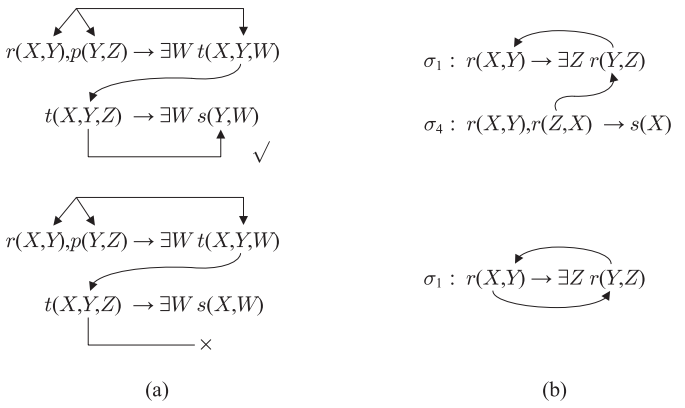

Fig. 1. Sticky property and propagation step.

In the context of the present paper, a set of TGDs can be seen as a logic program (or, equivalently, as an ASP program without disjunction and negation) since existentially quantified variables in rule-heads are equivalent to appropriate functional terms; e.g., the TGD $r(X) \rightarrow \exists Y\ s(X, Y)$ is equivalent (for query answering purposes) with the rule $r(X) \rightarrow s(X, f(X))$. This is true since we do not consider negation in existential rules, and we operate under the certain semantics which means that it suffices to employ the chase algorithm that never unifies null values.

Relevant Classes of TGDs. Guarded TGDs, inspired by the guarded fragment of first-order logic (Andréka *et al.* 1998), were proposed in Calì *et al.* (2008). A TGD σ is *guarded* if there exists an atom $\underline{a} \in body(\sigma)$, called *guard*, which contains all the variables occurring in $body(\sigma)$; let guarded be the class of guarded TGDs (Calì *et al.* 2008). Guarded TGDs with exactly one body-atom are called *linear*, and the resulted class is denoted linear (Calì *et al.* 2012). For example, the TGD $r(X, Y), s(Y, X, Z) \rightarrow \exists W\, r(Z, W)$ is guarded, where $s(Y, X, Z)$ is the guard and $r(X, Y)$ is a side atom. The chase of a database w.r.t. a set of guarded TGDs has bounded treewidth. This property allows us to show that query answering under guarded TGDs is decidable (Calì *et al.* 2008). The complexity of guarded TGDs has been also investigated in (Calì *et al.* 2008; Calì *et al.* 2012); it is PTIME-complete in data complexity, NP-complete in case of fixed TGDs, EXPTIME-complete in case of bounded arity, and 2EXPTIME-complete in general.

The class of sticky sets of TGDs, denoted sticky, was proposed in Calì *et al.* (2012) with the aim of identifying an expressive class that allows for joins in rule-bodies, which are expressible only via non-guarded rules. The key idea underlying stickiness is to ensure that, during the chase, terms which are associated with body-variables that appear more than once (i.e., join variables) always are propagated (or "stick") to the inferred atoms; this is illustrated in Figure 1(a). In particular, stickiness guarantees that the chase enjoys the so-called *sticky property* (Calì *et al.* 2012); for details see the online appendix. The definition of sticky sets of TGDs hinges on a variable-marking procedure called SMarking. For notational convenience, given a TGD σ, an atom $\underline{a} \in head(\sigma)$, and a universally quantified variable V of σ, $pos(\sigma, \underline{a}, V)$ is the set of positions in $\underline{a}$ at which V occurs. SMarking(Σ) is constructed

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

as follows. First, we apply on Σ the *initial marking* step: for each $\sigma \in \Sigma$, and for each variable $V \in var(body(\sigma))$, if there exists an atom $\underline{a} \in head(\sigma)$ such that $V \notin var(\underline{a})$, then each occurrence of V in $body(\sigma)$ is marked. $\mathsf{SMarking}(\Sigma)$ is obtained by applying exhaustively on Σ the *propagation* step: for each pair $\langle \sigma, \sigma' \rangle \in \Sigma \times \Sigma$, for each atom $\underline{a} \in head(\sigma)$, and for each universally quantified variable V of $var(\underline{a})$, if there exists an atom $\underline{b} \in body(\sigma')$ in which a marked variable occurs at each position of $pos(\sigma, \underline{a}, V)$, then each occurrence of V in $body(\sigma)$ is marked.

Example 2.1
Consider the set Σ constituted by $\sigma_1 : r(X, Y) \rightarrow \exists Z\, r(Y, Z)$, $\sigma_2 : r(X, Y) \rightarrow s(X)$, $\sigma_3 : s(X), s(Y) \rightarrow p(X, Y)$ and $\sigma_4 : r(X, Y), r(Z, X) \rightarrow s(X)$. By applying the initial marking (resp., propagation) step, the body-variables of Σ are marked with a cap (resp., double-cap) as follows: $\sigma_1 : r(\hat{X}, \hat{\hat{Y}}) \rightarrow \exists Z\, r(Y, Z)$, $\sigma_2 : r(X, \hat{Y}) \rightarrow s(X)$, $\sigma_3 : s(X), s(Y) \rightarrow p(X, Y)$ and $\sigma_4 : r(X, \hat{Y}), r(\hat{Z}, X) \rightarrow s(X)$. Figure 1(b) depicts the two ways of propagating the marking to the body-variable Y of σ_1. ∎

A set Σ of TGDs is *sticky* if, for every $\sigma \in \mathsf{SMarking}(\Sigma)$, each marked variable in $body(\sigma)$ appears only once. Observe that the set Σ given in Example 2.1 is sticky. Although stickiness does not ensure the finiteness of the treewidth of the chase, guarantees the termination of backward (resolution-based) chaining procedures, which in turn implies decidability of query answering. Query answering under sticky sets of TGDs is in AC_0 in data complexity, NP-complete in case of fixed TGDs, and EXPTIME-complete in combined complexity (Calì *et al.* 2012).

3 Tameness

We study the problem of joining guardedness and stickiness. At first glance, it may seem it could be sufficient to consider the union of guarded and sticky.

Definition 3.1 (*Union of Classes*)
Let $\mathfrak{C}_1$ and $\mathfrak{C}_2$ be arbitrary classes of TGDs. A set Σ of TGDs belongs to the *union* of $\mathfrak{C}_1$ and $\mathfrak{C}_2$, denoted $\mathfrak{C}_1|\mathfrak{C}_2$, if there exists a partition $\{\Sigma_1, \Sigma_2\}$ of Σ, where $\Sigma_1 \in \mathfrak{C}_1$ and $\Sigma_2 \in \mathfrak{C}_2$. Let $P_{\mathfrak{C}_1|\mathfrak{C}_2}(\Sigma)$ be the set of all possible such partitions of Σ. ∎

To avoid confusions, if $\{\Sigma_1, \Sigma_2\} \in P_{\mathfrak{C}_1|\mathfrak{C}_2}(\Sigma)$, for some arbitrary set Σ of TGDs, then $\Sigma_1 \in \mathfrak{C}_1$ and $\Sigma_2 \in \mathfrak{C}_2$, i.e., we first write the set of $\mathfrak{C}_1$ and then the set of $\mathfrak{C}_2$. Notice that, by definition, the classes $\mathfrak{C}_1|\mathfrak{C}_2$ and $\mathfrak{C}_2|\mathfrak{C}_1$ coincide.

Example 3.1
Let Σ be the set of TGDs constituted by $\sigma_1 : r(X, X, Y) \rightarrow \exists Z \exists W\, r(Y, Y, Z), s(Z), q(W)$, $\sigma_2 : t(X, Y) \rightarrow \exists Z\, r(Y, Z), p(Z)$, $\sigma_3 : s(X), q(Y) \rightarrow u(X, Y)$ and $\sigma_4 : p(X), p(Y) \rightarrow v(X, Y)$. Clearly, $P_{\mathsf{guarded}|\mathsf{sticky}}(\Sigma) = \{\{\{\sigma_1, \sigma_2\}, \{\sigma_3, \sigma_4\}\}, \{\{\sigma_1\}, \{\sigma_2, \sigma_3, \sigma_4\}\}\}$, and thus $\Sigma \in \mathsf{guarded}|\mathsf{sticky}$. ∎

As already thoroughly discussed in Baget *et al.* (2011), the union of two decidable classes, in general, leads to undecidability of query answering. This holds also for linear|sticky, even when further restrictions are imposed; for the proof see the online appendix.

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

Theorem 3.1
BCQ answering is undecidable under: (1) linear|sticky, even for a single linear TGD and a single sticky TGD, and (2) linear|sticky, even for sticky rules where each variable occurs only once.

The above result demonstrates the need of suggesting a class somewhat more restrictive than guarded|sticky. To tame the interaction between guarded and sticky rules it suffices to guarantee that none of the sticky rules "feeds" the guard atom of a guarded TGD during the construction of the chase. In other words, whenever a guarded rule σ is applied with homomorphism h, then its guard must be mapped by h into an atom obtained from a guarded rule. However, each other atom of $body(\sigma)$ can be mapped by h into an atom obtained from either a guarded or a sticky rule. To formalize this condition we need the so-called guard function.

Definition 3.2 (*Guard Function*)
Consider a set $\Sigma \in$ guarded. A *guard function* of Σ is a function $g : \Sigma \rightarrow \cup_{\sigma\in\Sigma} body(\sigma)$, where $g(\sigma) \in body(\sigma)$ and $var(g(\sigma)) = var(body(\sigma))$, for each $\sigma \in \Sigma$. Let $Guard(\Sigma)$ be the set of all possible guard functions of Σ. ■

We are now ready to give the formal definition of tameness.

Definition 3.3 (*Tameness*)
A set $\Sigma \in$ guarded|sticky is called *tame* if there exists $\{\Sigma_g, \Sigma_s\} \in P_{\text{guarded|sticky}}(\Sigma)$ for which the following condition holds: for each database D, and for each (possibly infinite) sequence of TGD chase steps $I_i\langle\sigma_i, h_i\rangle I_{i+1}$, where $i \geqslant 0$ and $I_0 = D$, there exists a guard function $g \in Guard(\Sigma_g)$ such that, for each $k > 0$ where $\sigma_k \in \Sigma_g$, if $\ell \in \{0, \ldots, k-1\}$ is the (unique) integer such that $h_k(g(\sigma_k)) \in I_{\ell+1} \setminus I_\ell$, then $\sigma_\ell \in \Sigma_g$. ■

Clearly, the tameness condition is not syntactic since it depends on the chase, and it is at the same level of abstraction as the previously mentioned classes fes, bts, fus and ps. However, if we force that none of the predicates that appear in the head of a sticky rule is used as the predicate of a guard atom, then we get a sufficient syntactic condition.

Definition 3.4 (*Predicate-tameness*)
A set $\Sigma \in$ guarded|sticky is called *predicate-tame* if there exists $\{\Sigma_g, \Sigma_s\} \in P_{\text{guarded|sticky}}(\Sigma)$ for which the following condition holds: there exists a guard function $g \in Guard(\Sigma_g)$ such that, for each $\sigma \in \Sigma_s$, there is no $\sigma' \in \Sigma_g$ for which $pred(g(\sigma')) \in pred(head(\sigma))$. ■

The set given in Example 1.1 is predicate-tame. It is easy to verify that predicate-tameness implies tameness. Surprisingly, even if the number of partitions of $P_{\text{guarded|sticky}}(\Sigma)$ is exponential in the worst-case, predicate-tameness can be checked in polynomial time; see the online appendix.

Proposition 3.1
The problem of deciding whether a set $\Sigma \in$ guarded|sticky is predicate-tame is in PTIME.

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

The predicate-tameness condition can be relaxed in several ways. For example, we can exploit the notion of the dependency graph (Fagin *et al.* 2005); let us illustrate this via an example. Consider the set Σ constituted by the TGDs $\sigma_1 : r(X,Y,Z), s(X) \rightarrow \exists W\, s(W)$ and $\sigma_2 : s(X), s(Y) \rightarrow \exists Z\, r(Z,X,Y)$; note that σ_1 is guarded while σ_2 is sticky. Σ is not predicate-tame, but it is tame since the join operation (over X) in $body(\sigma_1)$ cannot be satisfied, whatever the input database is. This holds since the null value generated at position $r[1]$ by applying σ_2, cannot be propagated to position $s[1]$. Thus, there is no way for an atom generated by applying σ_2 to "feed" the guard of σ_1. This can be detected by inspecting the dependency graph of Σ. For brevity, in the rest of the paper, whenever we say a tame set of TGDs we mean a tame guarded|sticky set of TGDs.

4 Querying the tame fragment

We study the problem of query answering under tame sets of TGDs. The fact that a sticky rule may "feed" a side atom of a guarded rule destroys the main properties of the chase ensured by guardedness, and therefore the existing algorithms for guarded TGDs are inappropriate in our case. Thus, we had to look for new decision procedures beyond the state of the art. A key notion employed in the guarded case is the *type* of an atom $\underline{a} \in chase(D,\Sigma)$, defined as $type(\underline{a},D,\Sigma) = \{\underline{b} \in chase(D,\Sigma) \mid terms(\underline{b}) \subseteq terms(\underline{a})\}$. The central importance of type in this case is that the subtree of the *guarded chase forest* of D and Σ, that is, the structure obtained from $chase(D,\Sigma)$ by keeping only the guards and their children, rooted at $\underline{a}$ is determined by $type(\underline{a},D,\Sigma)$ (modulo renaming of nulls) (Calì *et al.* 2008). (In the sequel, the atoms that form the guarded chase forest rooted at $\underline{a}$ are denoted by $reach_g(\underline{a},D,\Sigma)$.) This fact is at the basis of the existing algorithms for guarded rules. Unfortunately, due to the presence of sticky rules, tame sets of TGDs do not enjoy the above property as shown by the following example.

Example 4.1
Let Σ be the tame set of TGDs:

$\sigma_1 : p_1(X) \rightarrow \exists Z\, p_4(X,Z)$
$\sigma_2 : p_4(X,Y) \rightarrow \exists Z \exists W\, p_5(X,Y,Z), p_6(Y,W,X)$
$\sigma_3 : p_5(X,Y,Z) \rightarrow p_7(X,Y,Z)$
$\sigma_4 : p_5(X,Y,Z) \rightarrow s(X)$
$\sigma_5 : p_0(X,Y), s(Y) \rightarrow r(Y)$
$\sigma_6 : r(X), p_5(X,Y,Z), p_9(X,Y) \rightarrow \exists W\, p_{11}(Y,W)$
$\sigma_7 : p_2(X), p_6(Y,Z,W), p_7(W,Y,V) \rightarrow p_8(W,Y,Z,X)$
$\sigma_8 : p_8(X,Y,Z,W), p_3(V,X) \rightarrow p_9(X,Y)$
$\sigma_9 : p_8(X,Y,Z,W), p_3(W,X) \rightarrow p_{10}(X,Y,Z,W),$

where $\Sigma_s = \{\sigma_7, \sigma_8, \sigma_9\}$ and $\Sigma_g = \Sigma \setminus \Sigma_s$, and $D = \{p_0(d,b), p_1(b), p_2(c), p_3(c,b)\}$. The chase of D and Σ is depicted in Figure 2. Bold and continuous arrows denote guarded and sticky chase derivations, respectively; dashed arrows denote the contribution from side atoms in guarded derivations only. Notice that $p_{11}(z_1,z_4)$ is obtained from σ_6, the application of which involves $p_9(b,z_1)$ obtained from the sticky rule σ_8, that is triggered due to $p_3(c,b) \notin type(\underline{a},D,\Sigma)$. ∎

Plan of Attack. Our goal is to extend the notion of type in such a way that the above key property for guarded TGDs holds also for tame sets of TGDs. In particular, we associate to an atom $\underline{a}$ the so-called *active type* of $\underline{a}$ w.r.t. a database D

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

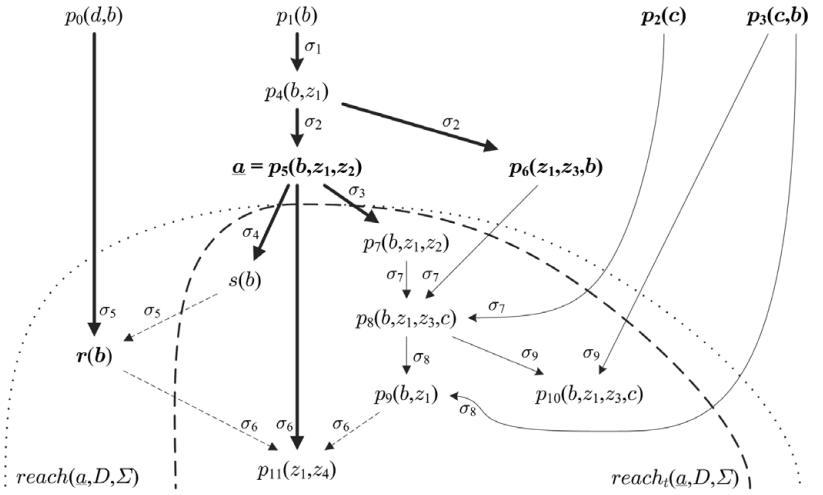

Fig. 2. The instance $chase(D,\Sigma)$ and the active type of an atom.

and tame set Σ of TGDs, denoted by $atype(\underline{a},D,\Sigma)$; for example, the active type of the atom $p_5(b,z_1,z_2)$ in Figure 2 is constituted by the five boldfaced atoms, which are outside the dashed boundary. This extended notion of type allows us to determine an interesting superset of $reach_g(\underline{a},D,\Sigma)$, denoted by $reach_t(\underline{a},D,\Sigma)$; see the atoms inside the dashed boundary in Figure 2. Roughly, $reach_t(\underline{a},D,\Sigma)$ is the set of atoms of the chase that depend directly or indirectly on $\underline{a}$, and that they are generated either by some guarded rule involving a guard atom which also belongs to $reach_t(\underline{a},D,\Sigma)$ or by some sticky rule. Unfortunately, the active type is in general infinite. However, what we really need is a relevant subset of $reach_t(\underline{a},D,\Sigma)$ constituted by the atoms generated by guarded rules ($p_7(b,z_1,z_2)$, $s(b)$ and $p_{11}(z_1,z_4)$ in Figure 2) plus those that are used as side atoms, even if they are not generated by guarded rules ($p_9(b,z_1)$ in Figure 2); we refer to this set as $reach_{gs}(\underline{a},D,\Sigma)$. Interestingly, if Σ consists of guarded rules, $reach_g(\underline{a},D,\Sigma)$, $reach_t(\underline{a},D,\Sigma)$ and $reach_{gs}(\underline{a},D,\Sigma)$ coincide, while $atype(\underline{a},D,\Sigma) \subseteq type(\underline{a},D,\Sigma)$. However, in case Σ is a tame set, to construct $reach_{gs}(\underline{a},D,\Sigma)$ it is sufficient to consider a finite subset of the active type of $\underline{a}$ which, asymptotically, has the same size as the type of $\underline{a}$. Notice that the set of atoms that depend on $\underline{a}$, denoted as $reach(\underline{a},D,\Sigma)$, is in general a superset of $reach_t(\underline{a},D,\Sigma)$; atoms inside the dotted boundary in Figure 2. In fact, the atoms of $reach(\underline{a},D,\Sigma) \setminus reach_t(\underline{a},D,\Sigma)$ ($r(b)$ in Figure 2) are generated by guarded rules and the guard atoms that are involved in their generation ($p_0(d,b)$ in Figure 2) do not depend on $\underline{a}$.

Technical Results. Let us now formalize the above intuitive description. Fix an instance I, a tame set Σ, and an atom $\underline{a} \in chase(I,\Sigma)$. If a pair $\langle \underline{a},\underline{b} \rangle$ belongs to $CR^+[I,\Sigma]$, then $\underline{b}$ is obtained due to some chase derivation that involves $\underline{a}$, and we say that $\underline{b}$ is *reachable* from $\underline{a}$. Accordingly, we define $reach(\underline{a},I,\Sigma) = \{\underline{b} \mid \langle \underline{a},\underline{b} \rangle \in CR^+[I,\Sigma]\}$. Let $\{\Sigma_g,\Sigma_s\} \in P_{\mathsf{guarded|sticky}}(\Sigma)$ be the partition of Σ, and $g \in Guard(\Sigma_g)$ be a guard function provided by Definition 3.3. For an atom $\underline{b} \in reach(\underline{a},I,\Sigma)$ obtained from some $\sigma \in \Sigma$ with homomorphism h, we denote by $parent(\underline{b},I,\Sigma)$

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

the set $\{h(g(\sigma))\}$ (resp., $h(body(\sigma))$) if $\sigma \in \Sigma_g$ (resp., $\sigma \in \Sigma_s$). Notice that, for each $\underline{c} \in parent(\underline{b}, I, \Sigma)$, $\langle \underline{c}, \underline{b} \rangle \in CR[I, \Sigma]$. Intuitively, if $\underline{b}$ is obtained from a guarded rule, then its parent-set is a singleton containing the guard atom that has been involved in its generation; otherwise, its parent-set contains all the atoms involved in its generation. For example, in Figure 2, $parent(p_{11}(z_1, z_4), D, \Sigma)$ is the singleton $\{p_5(b, z_1, z_2)\}$, while $parent(p_9(b, z_1), D, \Sigma)$ is the set $\{p_3(c, b), p_8(b, z_1, z_3, c)\}$. We are now ready to define the set $reach_t(\underline{a}, I, \Sigma)$ which is obtained from $reach(\underline{a}, I, \Sigma)$ by eliminating every atom that is generated by a rule of Σ_g, but its parent is not reachable from $\underline{a}$. In fact, an atom belongs to $reach(\underline{a}, I, \Sigma) \setminus reach_t(\underline{a}, I, \Sigma)$ because some side atom that has been involved in its generation is reachable from $\underline{a}$ (see atom $r(b)$ in Figure 2).

Definition 4.1 (*Tame Reachability*)
The set of atoms $reach_t(\underline{a}, I, \Sigma) \subseteq reach(\underline{a}, I, \Sigma)$ is inductively defined as $\{\underline{b} \mid \underline{a} \in parent(\underline{b}, I, \Sigma)\} \cup \{\underline{b} \mid parent(\underline{b}, I, \Sigma) \cap reach_t(\underline{a}, I, \Sigma) \neq \emptyset\}$. ∎

Recall that in case Σ_s is empty the set $reach_t(\underline{a}, I, \Sigma)$ coincides with $reach_g(\underline{a}, I, \Sigma)$. We now refine, and at the same time extend, the notion of type by highlighting the atoms which are not in $reach_t(\underline{a}, I, \Sigma)$, but are directly involved in its construction; this set is called the *active type* of $\underline{a}$.

Definition 4.2 (*Active Type*)
An atom $\underline{c} \in chase(I, \Sigma)$ belongs to the *active type* of $\underline{a}$ w.r.t. I and Σ, denoted as $atype(\underline{a}, I, \Sigma)$, if $\underline{c} \notin reach_t(\underline{a}, I, \Sigma)$ and also there exists $\underline{b} \in reach_t(\underline{a}, I, \Sigma)$ such that $\langle \underline{c}, \underline{b} \rangle \in CR[I, \Sigma]$. ∎

In Figure 2, the boldfaced atoms are in the active type of $p_5(b, z_1, z_2)$ since they have outgoing arrows crossing the dashed boundary of $reach_t(\underline{a}, D, \Sigma)$. Notice that an atom of $reach(\underline{a}, I, \Sigma) \setminus reach_t(\underline{a}, I, \Sigma)$ may belong to the active type of $\underline{a}$ w.r.t I and Σ (see $r(b)$ in Figure 2). By definition 4.2, it is easy to see that $reach_t(\underline{a}, I, \Sigma) \simeq_{terms(\underline{a})} reach_t(\underline{a}, atype(\underline{a}, I, \Sigma), \Sigma)$ which means that we can identify $reach_t(\underline{a}, I, \Sigma)$ by exploiting the active type of $\underline{a}$. Observe that if $\Sigma_s = \emptyset$, then $atype(\underline{a}, I, \Sigma) \subseteq type(\underline{a}, I, \Sigma)$, and in case of linear rules, $atype(\underline{a}, I, \Sigma) = \{\underline{a}\}$. Unfortunately, $atype(\underline{a}, I, \Sigma)$ is in general infinite. However, starting from an atom $\underline{a}$ obtained due to a guarded rule, what we really need in order to design a query answering algorithm for tame TGDs is to generate (as in the guarded case) *(i)* the atoms of $reach_t(\underline{a}, I, \Sigma)$ which are directly involved in the "guarded chase derivations", i.e., those that are generated by rules of Σ_g (atoms $p_7(b, z_1, z_2)$, $s(b)$ and $p_{11}(z_1, z_4)$ in Figure 2), and *(ii)* the atoms which have been used as side atoms during the applications of rules of Σ_g (atom $p_9(b, z_1)$ in Figure 2). We denote this set as $reach_{gs}(\underline{a}, I, \Sigma)$.

Definition 4.3 (*Guard-side Reachability*)
The set of atoms $reach_{gs}(\underline{a}, I, \Sigma)$ is inductively defined as follows: (1) if $\underline{c} \in reach_t(\underline{a}, I, \Sigma)$ is generated by a rule of Σ_g and $parent(\underline{c}, I, \Sigma) = \{\underline{a}\}$, then $\underline{c} \in reach_{gs}(\underline{a}, I, \Sigma)$, (2) if $\underline{c} \in reach_t(\underline{a}, I, \Sigma)$ is generated by a rule of Σ_g and $parent(\underline{c}, I, \Sigma) \subseteq reach_{gs}(\underline{a}, I, \Sigma)$, then $\underline{c} \in reach_{gs}(\underline{a}, I, \Sigma)$, and (3) if $\underline{c} \in reach_{gs}(\underline{a}, I, \Sigma)$ is generated

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

by a rule of Σ_g, and there exists $\underline{b} \in reach_t(\underline{a}, I, \Sigma)$ such that $\langle \underline{b}, \underline{c} \rangle \in CR[I, \Sigma]$, then $\underline{b} \in reach_{gs}(\underline{a}, I, \Sigma)$. ∎

Let $S_{in}(\underline{a}, I, \Sigma) = \bigcup_{\underline{b} \in reach_{gs}(\underline{a}, I, \Sigma)} terms(\underline{b})$ and $S_{out}(\underline{a}, I, \Sigma) = \bigcup_{\underline{b} \in atype(\underline{a}, I, \Sigma)} terms(\underline{b})$. In other words, $S_{in}(\underline{a}, I, \Sigma)$ collects the terms of $reach_{gs}(\underline{a}, I, \Sigma)$, while $S_{out}(\underline{a}, I, \Sigma)$ collects the terms of $atype(\underline{a}, I, \Sigma)$. For brevity, we refer to the above sets by S_{in} and S_{out}, respectively. The next lemma shows that the terms occurring in the active type of $\underline{a}$ but not in $\underline{a}$, do not appear in $reach_{gs}(\underline{a}, I, \Sigma)$.

Lemma 4.1
It holds that, $S_{in} \cap S_{out} \subseteq terms(\underline{a})$.

We are now ready to establish that $reach_{gs}(\underline{a}, I, \Sigma)$ can be determined by a finite subset of $atype(\underline{a}, I, \Sigma)$. The key idea underlying this result is that $reach_{gs}(\underline{a}, I, \Sigma)$ can be obtained by considering as a database some representative atoms of $atype(\underline{a}, I, \Sigma)$, which in turn are obtained by keeping unaltered the terms of $\underline{a}$ and replacing all the other terms occurring in $atype(\underline{a}, I, \Sigma)$ by some special character. For instance, in Example 4.1, it is easy to verify that $reach_{gs}(\underline{a}, D, \Sigma) = \{p_7(b, z_1, z_2), s(b), p_9(b, z_1), p_{11}(z_1, z_4)\}$ can be generated by applying some chase steps staring from a database constituted, apart from $r(b)$ and $p_5(b, z_1, z_2)$, also by the representative atoms $p_6(z_1, \star, b)$, $p_2(\star)$ and $p_3(\star, b)$, where $\star$ is a special "don't care" character.

Proposition 4.1
There exists a finite set $T \subseteq atype(\underline{a}, I, \Sigma)$ such that $reach_{gs}(\underline{a}, I, \Sigma) \simeq_{terms(\underline{a})} reach_{gs}(\underline{a}, T, \Sigma)$.

Proof
We identify the set T as follows: (1) construct $T_\star$ from $atype(\underline{a}, I, \Sigma)$ by replacing every term of $S_{out} \setminus terms(\underline{a})$ with the special symbol $\star$, (2) for each $\underline{b} \in T_\star$, if $\star$ appears in $\underline{b}$ at position i, then replace this occurrence of $\star$ in $\underline{b}$ with the symbol $\star_{\underline{b},i}$, and (3) for each $\underline{b} \in T_\star$, add to T an atom $\underline{c}$ that is arbitrarily chosen from $atype(\underline{a}, I, \Sigma)$ in such a way that there exists a homomorphism h such that $h(\underline{b}) = \underline{c}$. To show that T is finite it suffices to show that $T_\star$ is finite. By construction, $T_\star$ is a subset of the atoms that can be formed with predicates from the underlying schema and terms from $terms(\underline{a}) \cup \{\star\}$; thus, $T_\star$ is finite. Clearly, the second step does not alter the size of $T_\star$. Let us now show that $reach_{gs}(\underline{a}, I, \Sigma) \simeq_{terms(\underline{a})} reach_{gs}(\underline{a}, T, \Sigma)$. By Definitions 4.1 and 4.2, each atom of $atype(\underline{a}, I, \Sigma) \setminus type(\underline{a}, I, \Sigma)$ is involved in the generation of some atom of $reach_t(\underline{a}, I, \Sigma)$ only via sticky rules. (This is the kind of atoms $p_2(c)$, $p_3(c, b)$, and $p_6(z_1, z_3, b)$ in Figure 2, which have outgoing arrows crossing the boundary of $reach_t(\underline{a}, D, \Sigma)$ and that are labeled by sticky rules.) By stickiness, each term of $S_{out} \setminus terms(\underline{a})$ which is lost in some chase derivation that reaches an atom of $reach_{gs}(\underline{a}, I, \Sigma)$ cannot be in any join, and also, by Lemma 4.1, does not appear in S_{in}. But these are exactly the terms occurring in $atype(\underline{a}, I, \Sigma)$ that have been replaced by $\star$ in the construction of $T_\star$. Thus, T is the correct witness of $atype(\underline{a}, I, \Sigma)$ for the generation of the atoms that are used as side atoms in guarded chase derivation starting from $\underline{a}$. □

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

Input: An instance $\langle q, D, \Sigma \rangle$ of query answering.

1 Guess the following (chase) structures, and universally goto steps 2, 5 and 13;
  A homomorphism $h : terms(q) \to \Gamma \cup \Gamma_N$, and assign $h(q)$ to Q and $terms(Q) \cap \Gamma_N$ to N;
  A partition $\{N_g, N_s\}$ of N;
  A poset $P = \langle N_g, \dot{\longrightarrow} \rangle$, where each minimal element is the root of a rooted tree;
  A pair $\langle a_z, T(a_z) \rangle$, for each $z \in N_g$, and an atom a_y, for each $y \in N_s$;
2 Universally select every minimal element z of P and goto next step; `//guarded resolution steps`
3 Guess a TGD $\sigma \in \Sigma_g$ that admits an MGU θ of a_z and $head(\sigma)$;
4 Assign $T(a_z) \cup \theta(body(\sigma))$ to q, and goto step 1;
5 Universally select every $z \dot{\longrightarrow} \tilde{z} \in P$ and goto next step; `//guarded chase steps`
6 Let $cjump = 6$; if $a_z \simeq_{N_g} a_{\tilde{z}}$, then: If $T(a_z) \simeq_{N_g} T(a_{\tilde{z}})$, then *accept*, else *reject*;
7 Guess $\sigma \in \Sigma_g$, with g being its guard atom, that admits a homomorphism h such that $h(g) = a_z$;
8 Apply a chase step with σ and h, generate b, and guess a (finite) active type $T(b)$ for b;
9 Assign $(h(body(\sigma)) \cup T(b)) \setminus T(a_z)$ to Q;
10 Universally goto steps 11 and 13;
11 Assign $T(b)$ to $T(a_z)$, assign b to a_z, and assign the fresh null of b (if any) to z;
12 Goto step *cjump*;
13 Universally select every $a \in Q \setminus \{a_z \mid z \in N_g\}$ and goto next step; `//hybrid steps`
14 Let $A = \bigcup_{z \in N_g} T(a_z)$ denote the union of all the current (finite) active types;
15 If there is $c \in D \cup A$ s.t. $a \simeq_N c$, then *accept*, else nondeterministically goto step 16 or 19;
16 Guess a TGD $\sigma \in \Sigma_s$ that admits an MGU θ of a and $head(\sigma)$; `//sticky resolution steps`
17 If σ contains an $\exists$-quantified variable Y and $a \not\simeq_N a_{\theta(Y)}$, then *reject*;
18 Assign $\theta(body(\sigma))$ to Q, and goto step 13;
19 If a contains a null of N_s or two nulls of N_g that are uncomparable w.r.t. $\dot{\longrightarrow}$, then *reject*;
20 Let z be the greatest element of the chain $\langle terms(a) \cap N_g, \dot{\longrightarrow} \rangle$; `//guarded chase steps`
21 Let $cjump = 21$; if $a_z \simeq_{N_g} a$, then *accept*, else goto step 7;

Fig. 3. The alternating algorithm TameQAns.

The Algorithm TameQAns. We assume normalized sets of TGDs, where each rule has a single head-atom which contains only one occurrence of an existentially quantified variable; see the online appendix (Lemma C.1). Before presenting our algorithm, let us highlight, with the aid of Figure 3, the atoms of the chase that are crucial for deciding whether a query is entailed, and also how they are connected by the chase relation. Consider a database D, a BCQ q, and a tame set Σ, for which there exists a homomorphism h such that $h(q) \subseteq chase(D, \Sigma)$. Figure 3 depicts a segment of the chase of D and Σ; differently from Figure 2, each arrow may represent more than one chase derivation. In fact, bold arrows denote guarded chase derivations, while normal arrows denote sticky ones. Notice that between $g_4(z_4, z_3, \ldots)$ and $s_5(z_5, z_4)$, and also between $g_5(z_5, z_3, z_1, \ldots)$ and $s_4(y_4, z_5, z_7, z_6)$, we have a bold arrow followed by a normal one. This means that a guarded chase derivation is followed by a sticky one. The atoms at the bottom part form the set $h(q)$, while a null x is in boldface in an atom $\underline{a}$ if x is invented in $\underline{a}$; we refer to $\underline{a}$ by $\underline{a}_x$.

Let $\{N_g, N_s\}$ be the partition of $terms(h(q)) \setminus \Gamma$ such that N_g (resp., N_s) are the nulls introduced by guarded (resp., sticky) rules; clearly, $N_g = \{z_1, \ldots, z_8\}$ and $N_s = \{y_1, \ldots, y_4\}$. First, observe that by connecting the nulls of N_g with arrows of the form $\dot{\longrightarrow}$, we obtain a poset where each minimal element (z_1 and z_6) is the root of a rooted tree. In particular, each atom reached by a bold arrow contains the null invented in its direct predecessor (which is not necessarily its parent). Second,

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press

due to the presence of sticky rules, an atom $\underline{a} \in h(q)$ can mix nulls introduced by both guarded and sticky rules, or nulls introduced by guarded rules only, but occurring in atoms which do not appear on the same guarded chase derivation. Our alternating algorithm TameQAns is shown in Figure 3; for a detailed description of the algorithm see the online appendix.

By exploiting Proposition 4.1 and the sticky property, we can show via an inductive argument that indeed the algorithm TameQAns is sound and complete. It is also possible to show that, at each step of the computation, our algorithm needs exponential space in general, polynomial space in case of bounded arity, and logarithmic space if the query and the set of TGDs are fixed. The next result follows since AEXPSPACE = 2EXPTIME, APSPACE = EXPTIME and ALOGSPACE = PTIME; the lower bounds are inherited from BCQ answering under guarded (Calì *et al.* 2008).

Theorem 4.1
BCQ answering under tame sets of TGDs is 2EXPTIME-complete in combined complexity, EXPTIME-complete in case of bounded arity, NP-complete in case of fixed TGDs, and PTIME-complete in data complexity.

5 Conclusions

There is a number of challenging open problems to be tackled. The first one is whether tame guarded|sticky is *finitely controllable*, i.e., query answering under arbitrary models (the problem of this paper), and query answering under finite models coincide. Such a result would be of high relevance in the context of classical databases where finite models are considered. Notice that both guarded and sticky are finitely controllable (Barany *et al.* 2010; Gogacz and Marcinkowski 2013). The second one concerns the extension of tame guarded|sticky with disjunction. Note that query answering under guarded-based disjunctive existential rules has been studied in depth the last years (Alviano *et al.* 2012; Gottlob *et al.* 2012; Bourhis *et al.* 2013). Finally, we would like to explore the possibility of reducing query answering under tame guarded|sticky to the problem of evaluating a pure Datalog query over a database. This will lead to a practical query answering algorithm; in fact, we are currently working on a DLV-based reasoner for ontological reasoning under existential rules, and the obtained algorithm will be part of it.

Acknowledgements

This research has received funding from ERC Grant 246858 "DIADEM". Andreas Pieris also acknowledges the EPSRC Grant EP/G055114/1.

References

Alviano, M., Faber, W., Leone, N. and Manna, M. 2012. Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues. *Theory and Practice of Logic Programming 12,* 4-5, 701–718.

Andréka, H., van Benthem, J. and Németi, I. 1998. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic 27*, 217–274.

Baget, J.-F., Leclère, M., Mugnier, M.-L. and Salvat, E. 2011. On rules with existential variables: Walking the decidability line. *Artificial Intelligence 175,* 9-10, 1620–1654.

Barany, V., Gottlob, G. and Otto, M. 2010. Querying the guarded fragment. In *Proceedings of the 25th IEEE Symposium on Logic in Computer Science*, 1–10.

Beeri, C. and Vardi, M. Y. 1981. The implication problem for data dependencies. In *Proceedings of the 8th International Colloquium on Automata, Languages and Programming*, 73–85.

Bourhis, P., Morak, M. and Pieris, A. 2013. The impact of disjunction on query answering under guarded-based existential rules. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, To appear.

Calì, A., Gottlob, G. and Kifer, M. 2008. Taming the infinite chase: Query answering under expressive relational constraints. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning*, 70–80.

Calì, A., Gottlob, G. and Lukasiewicz, T. 2012. A general Datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics 14*, 57–83.

Calì, A., Gottlob, G. and Pieris, A. 2012. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence 193*, 87–128.

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

Chandra, A. K. and Merlin, P. M. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, 77–90.

Deutsch, A., Nash, A. and Remmel, J. B. 2008. The chase revisisted. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 149–158.

Fagin, R., Kolaitis, P. G., Miller, R. J. and Popa, L. 2005. Data exchange: Semantics and query answering. *Theoretical Computer Science 336,* 1, 89–124.

Gebser, M., Kaufmann, B., Neumann, A. and Schaub, T. 2007. Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 386–392.

Gogacz, T. and Marcinkowski, J. 2013. Converging to the chase - a tool for finite controllability. In *Proceedings of the 28th Annual IEEE Symposium on Logic in Computer Science*, To appear.

Gottlob, G. and Koch, C. 2004. Monadic Datalog and the expressive power of languages for web information extraction. *Journal of the ACM 51,* 1, 74–113.

Gottlob, G., Manna, M., Morak, M. and Pieris, A. 2012. On the complexity of ontological reasoning under disjunctive existential rules. In *Proceedings of the 37th International Symposium Mathematical Foundations of Computer Science*, 1–18.

Hajiyev, E., Verbaere, M. and de Moor, O. 2006. *codeQuest:* Scalable source code queries with Datalog. In *Proceedings of the 20th European Conference on Object-Oriented Programming*, 2–27.

Johnson, D. S. and Klug, A. C. 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences 28,* 1, 167–189.

Krötzsch, M. and Rudolph, S. 2011. Extending decidable existential rules by joining acyclicity and guardedness. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 963–968.

Leone, N., Manna, M., Terracina, G. and Veltri, P. 2012. Efficiently computable Datalog$^{\exists}$ programs. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning*.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562.

Lierler, Y. and Maratea, M. 2004. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, 346–350.

Maier, D., Mendelzon, A. O. and Sagiv, Y. 1979. Testing implications of data dependencies. *ACM Transactions on Database Systems 4,* 4, 455–469.

Marczak, W. R., Alvaro, P., Conway, N., Hellerstein, J. M. and Maier, D. 2012. Confluence analysis for distributed programs: A model-theoretic approach. In *Proceedings of Datalog 2.0*. 135–147.

Patel-Schneider, P. F. and Horrocks, I. 2007. A comparison of two modelling paradigms in the semantic web. *Journal of Web Semantics 5,* 4, 240–250.

https://doi.org/10.1017/S1471068413000550 Published online by Cambridge University Press