Chase Termination Beyond Polynomial Time

PHILIPP HANISCH, Knowledge-Based Systems Group, TU Dresden, Germany
MARKUS KRÖTZSCH, Knowledge-Based Systems Group, TU Dresden, Germany

The *chase* is a widely implemented approach to reason with tuple-generating dependencies (tgds), used in data exchange, data integration, and ontology-based query answering. However, it is merely a semi-decision procedure, which may fail to terminate. Many decidable conditions have been proposed for tgds to ensure chase termination, typically by forbidding some kind of "cycle" in the chase process. We propose a new criterion that explicitly allows some such cycles, and yet ensures termination of the standard chase under reasonable conditions. This leads to new decidable fragments of tgds that are not only syntactically more general but also strictly more expressive than the fragments defined by prior acyclicity conditions. Indeed, while known terminating fragments are restricted to PTime data complexity, our conditions yield decidable languages for any k-ExpTime. We further refine our syntactic conditions to obtain fragments of tgds for which an optimised chase procedure decides query entailment in PSpace or k-ExpSpace, respectively.

CCS Concepts: • **Theory of computation → Constraint and logic programming**; **Database constraints theory**; *Database query languages (principles)*; *Logic and databases.*

Additional Key Words and Phrases: tuple-generating dependencies, chase termination, restricted chase

ACM Reference Format:
Philipp Hanisch and Markus Krötzsch. 2024. Chase Termination Beyond Polynomial Time. *Proc. ACM Manag. Data* 2, 2 (PODS), Article 93 (May 2024), 17 pages. https://doi.org/10.1145/3651594

1 INTRODUCTION

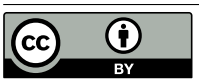
The *chase* [1, 6, 30] is an essential method for reasoning with constraints in databases, with application areas including data exchange [21], constraint implication [5], data cleansing [23], query optimization [2, 7, 33], query answering under constraints [12, 35], and ontological reasoning [4, 11]. The basis for this wide applicability is the chase's ability to compute a *universal model* for a set of constraints [18], which is either used directly (e.g., as a repaired database) or indirectly (e.g., for deciding query entailment).

However, universal models can be infinite, and chase termination is undecidable on a single database [5] as well as in its stricter *uniform* version over arbitrary databases [24, 26]. The root of this problem are a form of constraints known as *tuple-generating dependencies* (tgds) – or *existential rules*: Horn logic rules with existential quantification in conclusions –, since they may require additional domain elements (represented by *nulls*) to be satisfied.

A large body of research is devoted to finding decidable cases for which termination can be guaranteed, mainly by analysing the data flow, i.e., the propagation of nulls in the chase [13, 16, 19, 21, 28, 31].[1] Here we can distinguish graph-based abstractions, such as *weak acyclicity* [21], from materialisation-based approaches, such as MFA [16]. In general, decidable criteria are sufficient but

[1] As a notable exception, control flow is analysed in the *graph of rule dependencies* [4].

Authors' addresses: Philipp Hanisch, philipp.hanisch1@tu-dresden.de, Knowledge-Based Systems Group, TU Dresden, Dresden, Germany; Markus Krötzsch, markus.kroetzsch@tu-dresden.de, Knowledge-Based Systems Group, TU Dresden, Dresden, Germany.



This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2024 Copyright held by the owner/author(s).
ACM 2836-6573/2024/5-ART93
https://doi.org/10.1145/3651594

not necessary for termination, but recent breakthroughs established decidability of termination for the linear, guarded, and sticky classes of tgds [9, 10, 25, 34].

Meanwhile, another productive line of recent research clarified the computational power of the chase, characterising the query functions that can be expressed by conjunctive queries under tgd constraints. This expressive power is bounded by data complexity, but can be lower: famously, Datalog does not capture all queries in P, not even those *closed under homomorphisms*[2] [17]. Results are more satisfying for tuple-generating dependencies. Not only do arbitrary tgds capture the recursively enumerable homomorphism-closed queries [36], but, surprisingly, the decidable homomorphism-closed queries can be captured by tgds for which the standard (a.k.a. *restricted*) chase terminates [8]. In other words, the standard chase over tgds without any extensions is a universal computational paradigm for database query answering. This is highly encouraging since a majority of chase implementations already support this version of the chase procedure [3, 7, 23, 33, 35, 38], often with favourable performance [6].

Naturally, tractable data complexity is often desirable in database applications, and has therefore been the focus of many works in the area. Unfortunately, the potential of chase-based computation for more complex computations has meanwhile been neglected. To our knowledge, the only line of research where the chase was used for harder-than-P computations relies on a method for modelling finite sets with tgds [14]. Practical feasibility was shown for ExpTime-complete ontology-based query answering [14], set-terms in answer set programming [22], complex values in Datalog [32], and why-provenance [20]. In essence, all of these works are based on a single set of tgds for which uniform termination is shown directly. Known chase termination criteria fail for this case, since they only recognise queries in P. In fact, most criteria are even known to describe fragments that do not increase upon the expressive power of Datalog [29, 39], This limitation is common to all tgd sets on which the semi-oblivious chase uniformly terminates. We are aware of only one approach so far that studies the more complicated standard chase at all [13], but which also remains in P.

We tackle this challenge with a new method that extends the graph-based termination criterion of *joint acyclicity* [28] with new decidable conditions that allow for some kinds of cycles. These conditions are specific to the standard chase, and enforce that tgd applications within a cycle are eventually blocked, possibly only after (double) exponentially many loops. Our conditions detect the uniform termination of the set-modelling tgds of Carral et al. [14], but significantly extend upon this baseline: even a single strongly connected component in the data-flow graph can correspond to a 2-ExpTime-complete query (whereas Carral et al. deal with exponentially many sets).

Leveraging again the graph-based view, we can further analyse the data flows between cyclic strongly connected components to describe decidable fragments of arbitrary multi-exponential data complexity. Our resulting decidable fragment of *saturating tgds* therefore has non-elementary complexity. This is already very general, especially when considering that capturing all decidable homomorphism-closed queries can in principle only be achieved with tgd fragments that are not even recursively enumerable [8].

The structure of the data-flow graph enables us to compute more precise κ-ExpTime bounds for any saturating tgd set. Refining this analysis further, we then identify cases where the complexity of query answering drops to $(\kappa - 1)$-ExpSpace. In particular, we therefore obtain a decidable fragment of uniformly terminating tgd sets with PSpace-complete query entailment. To the best of our knowledge, this is the first such fragment.

All of our results establish uniform termination of the standard chase under all chase strategies that prioritise Datalog rules (Krötzsch et al. call this the *Datalog-first* strategy [29]). By a recent result, this is a stronger requirement than termination under *some* strategies [15]. As of today,

[2]Such queries are monotone, i.e., intuitively do not need negation (but cf. [37]).

however, all known termination criteria imply Datalog-first termination, and preferring Datalog rules is also a common heuristic in practice.

In summary, our main contributions are as follows:

- In Section 3, we refine the *dependency graph* of Krötzsch and Rudolph [28] to capture data flow in more detail.
- In Section 4, we study the propagation of inferences between nulls related to strongly connected components in the extended dependency graph. We find conditions that suffice to prevent infinite repetitions of inference cycles, define the language of *saturating tgds*, and show uniform standard chase termination for this fragment.
- In Section 5, we analyse the exact complexity (and size) of the standard chase over saturating sets by assigning *ranks* to strongly connected components in the extended dependency graph. We differentiate the case of single and double exponential complexity for individual components, and we establish matching lower bounds to show that query entailment is κ-ExpTime-complete for tgd sets of rank κ.
- In Section 6, we describe conditions that reduce query entailment for tgd sets of rank κ to $(\kappa - 1)$-ExpSpace. To this end, we discover a tree-like structure within the chase, and we define a syntactic condition called *path guardedness* that allows us to use an optimised chase procedure. Again, we establish matching lower bounds.

Detailed proofs are found online [27].

2 PRELIMINARIES

We consider a signature based on mutually disjoint, countably infinite sets of *constants* $\mathbf{C}$, *variables* $\mathbf{V}$, *predicates* $\mathbf{P}$, and *nulls* $\mathbf{N}$. Each predicate name $p \in \mathbf{P}$ has an *arity* $\mathrm{ar}(p) \geq 0$. *Terms* are elements of $\mathbf{V} \cup \mathbf{N} \cup \mathbf{C}$. We use $\boldsymbol{t}$ to denote a list $t_1, \ldots, t_{|\boldsymbol{t}|}$ of terms, and similarly for special types of terms. An *atom* is an expression $p(\boldsymbol{t})$ with $p \in \mathbf{P}$, $\boldsymbol{t}$ a list of terms, and $\mathrm{ar}(p) = |\boldsymbol{t}|$. An *interpretation* $\mathcal{I}$ is a set of atoms without variables. A *database* $\mathcal{D}$ is a finite interpretation without nulls, i.e., a finite set of *facts* (variable-free, null-free atoms). For an interpretation $\mathcal{I}$, we use $\mathbf{N}(\mathcal{I}) = \{n \in \mathbf{N} \mid p(\boldsymbol{t}) \in \mathcal{I}, n \in \boldsymbol{t}\}$ to denote the nulls used in $\mathcal{I}$.

Rules. A *tuple-generating dependency* (tgd) ρ is a formula

$$\rho = \forall \boldsymbol{x}, \boldsymbol{y}.\, B[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{v}.\, H[\boldsymbol{y}, \boldsymbol{v}], \tag{1}$$

where B and H are conjunctions of atoms using only terms from $\mathbf{C}$ or from the mutually disjoint lists of variables $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{v} \subseteq \mathbf{V}$. We call B the *body* (denoted $\mathrm{body}(\rho)$), H the *head* (denoted $\mathrm{head}(\rho)$), and $\boldsymbol{y}$ the *frontier* of ρ. We may treat conjunctions of atoms as sets, and we omit universal quantifiers in tgds. We require that all variables in $\boldsymbol{y}$ do really occur in B (*safety*). A tgd without existential quantifiers is a *Datalog rule*.

Renamings and Substitutions. Without loss of generality, we require that variables in tgd sets Σ are *renamed apart*, i.e., each variable $x \in \mathbf{V}$ in Σ is bound by a unique quantifier in a unique tgd $\rho_x \in \Sigma$. A *substitution* is a partial mapping $\sigma : \mathbf{V} \rightarrow \mathbf{C} \cup \mathbf{V} \cup \mathbf{N}$. As usual, $x\sigma = \sigma(x)$ if σ is defined on x, and $x\sigma = x$ otherwise. For a logical expression $\alpha[\boldsymbol{x}]$, the expression $\alpha\sigma$ is obtained by simultaneously replacing each variable x by $x\sigma$. Given a list of terms $\boldsymbol{t} = t_1, \ldots, t_{|\boldsymbol{x}|}$, we write $\alpha[\boldsymbol{x}/\boldsymbol{t}]$ for the expression $\alpha\{x_1 \mapsto t_1, \ldots, x_{|\boldsymbol{x}|} \mapsto t_{|\boldsymbol{x}|}\}$. If $\boldsymbol{x}$ is clear from the context and no confusion is likely, we write $\alpha[\boldsymbol{t}]$ for $\alpha[\boldsymbol{x}/\boldsymbol{t}]$.

Semantics. We consider a standard first-order semantics. A *match* of a tgd ρ as in (1) in an interpretation $\mathcal{I}$ is a substitution σ that maps $\boldsymbol{x} \cup \boldsymbol{y}$ to terms in $\mathcal{I}$, such that $B\sigma \subseteq \mathcal{I}$. A match is *satisfied* if it can be extended to a substitution σ' over $\boldsymbol{x} \cup \boldsymbol{y} \cup \boldsymbol{v}$ such that $H\sigma' \subseteq \mathcal{I}$. A tgd ρ is

satisfied by $\mathcal{I}$, written $\mathcal{I} \models \rho$, if all matches of ρ on $\mathcal{I}$ are satisfied. A fact α is satisfied in $\mathcal{I}$, written $\mathcal{I} \models \alpha$, if $\alpha \in \mathcal{I}$. Satisfaction extends to sets of tgds and facts as usual. A tgd or fact α is *entailed* by tgd set Σ and database $\mathcal{D}$ if $\mathcal{I} \models \alpha$ for all $\mathcal{I}$ with $\mathcal{I} \models \Sigma \cup \mathcal{D}$.

Reasoning with the Chase. An important reasoning task for tgds is conjunctive query (CQ) answering, which can further be reduced to the entailment of Boolean CQs (BCQs), which are formulas $\exists \boldsymbol{z}.Q[\boldsymbol{z}]$ with Q a conjunction of null-free atoms. This task is undecidable in general. A sound and complete (but not always terminating) class of reasoning procedures is the *chase*, which exists in many variants. We are interested in the *standard chase* (a.k.a. *restricted chase*) under *Datalog-first* strategies.

Definition 1. *A* (standard) chase sequence *for a database $\mathcal{D}$ and a tgd set Σ is a potentially infinite sequence of interpretations $\mathcal{D}^0, \mathcal{D}^1, \ldots$ such that*

(1) $\mathcal{D}^0 = \mathcal{D}$;
(2) for every $\mathcal{D}^{i+1}$ with $i \geq 0$, there is a match σ for some tgd $\rho = B[\boldsymbol{x}, \boldsymbol{y}] \to \exists \boldsymbol{v}.H[\boldsymbol{y}, \boldsymbol{v}] \in \Sigma$ in $\mathcal{D}^i$ such that both of the following hold true:
  (a) σ is an unsatisfied match in $\mathcal{D}^i$ (i.e., σ cannot be extended to a substitution σ^+ with $H\sigma^+ \subseteq \mathcal{D}^i$),
  (b) $\mathcal{D}^{i+1} = \mathcal{D}^i \cup H[\sigma^+(\boldsymbol{y}), \sigma^+(\boldsymbol{v})]$, where σ^+ is such that $\sigma^+(y) = \sigma(y)$ for all $y \in \boldsymbol{y}$, and for all $v \in \boldsymbol{v}$, $\sigma^+(v) \in \mathbf{N}$ is a distinct null not occurring in $\mathcal{D}^i$;
  we then say that ρ was applied *in step i, and we define $\mathrm{tgd}[i] := \rho$, $\sigma[i] := \sigma$, and $\sigma^+[i] := \sigma^+$;*
(3) if a tgd with existential variables is applied in step i, then $\mathcal{D}^i$ must satisfy all Datalog rules in Σ;
(4) if σ is a match for a tgd $\rho \in \Sigma$ and $\mathcal{D}^i$ ($i \geq 0$), then there is $j > i$ such that σ is satisfied in $\mathcal{D}^j$.

Item (3) *requires rule applications to follow a* Datalog-first *strategy, and item* (4) *ensures* fairness. *The* (standard) chase *for such a chase sequence then is* $\mathsf{chase}(\Sigma, \mathcal{D}) = \bigcup_{i \geq 0} \mathcal{D}^i$.

A BCQ q is entailed by Σ and $\mathcal{D}$ if and only if $\mathsf{chase}(\Sigma, \mathcal{D}) \models q$. If the chase terminates, this can be determined from the (finite) $\mathsf{chase}(\Sigma, \mathcal{D})$. Termination may depend on the chosen order of tgd applications. The Datalog-first strategy (3) is a common heuristic that tends to improve termination in practice, although one can construct examples where this is not the case [15]. Since Datalog rules can only be applied finitely many times, Datalog-first does not impair fairness (4).

3 THE LABELLED DEPENDENCY GRAPH

The *existential dependency graph* is used to analyse the data flow between existential variables in a tgd set [28]. In particular, a tgd set is *jointly acyclic* if this graph does not have cycles. In this section, we recall and slightly extend this approach to better suit our needs.

The following definition mostly follows Krötzsch and Rudolph [28], but adds variables as labels to the edges in the graph. Recall that we assume tgd sets to be renamed apart, so that variables x can be used to identify tgds ρ_x.

Definition 2. *Let Σ be a tgd set. A* predicate position *is a pair $\langle p, i \rangle \in \mathbf{P} \times \mathbb{N}$ with $1 \leq i \leq \mathrm{ar}(p)$. For a variable x in Σ, let Pos^B_x (resp. Pos^H_x) be the set of all predicate positions where x occurs in the body (resp. head) of its unique tgd $\rho_x \in \Sigma$.*

For an existential variable v in Σ, let Ω_v be the smallest set of positions such that (i) $\mathrm{Pos}^H_v \subseteq \Omega_v$, and (ii) for every universal variable x, $\mathrm{Pos}^B_x \subseteq \Omega_v$ implies $\mathrm{Pos}^H_x \subseteq \Omega_v$.

The labelled existential dependency graph $\mathsf{LXG}(\Sigma)$ *of Σ is a directed graph with the existentially quantified variables of Σ as its vertices and an edge $v \xrightarrow{y} w$ for every tgd $\rho_w \in \Sigma$ with a frontier variable y such that $\mathrm{Pos}^B_y \subseteq \Omega_v$.*

Example 3. *This running example illustrates several notions below, hence is not minimal. Consider a database $\mathcal{D}_{lvl} = \{first(1), last(\ell)\} \cup \{next(i, i+1) \mid 1 \leq i < \ell\}$, which defines a total strict order of*

"levels" $1, \dots, \ell$. Σ *consists of the following tgds with constants* F *(false) and* T *(true):*

$$first(z) \to lvl(F, z) \wedge lvl(T, z) \tag{2}$$

$$lvl(\bar{x}_1, z) \wedge lvl(\bar{x}_2, z) \to \exists v.cat(\bar{x}_1, \bar{x}_2, z, v) \tag{3}$$

$$cat(\bar{x}_1, \bar{x}_2, z, x) \to part(\bar{x}_1, x) \wedge part(\bar{x}_2, x) \tag{4}$$

$$cat(\bar{x}_1, \bar{x}_2, z, x) \wedge next(z, z_+) \to \exists \bar{w}.up(x, z_+, \bar{w}) \tag{5}$$

$$cat(\bar{x}_1, \bar{x}_2, z, x) \wedge next(z, z_+) \wedge up(x, z_+, \bar{x}) \to lvl(\bar{x}, z_+) \tag{6}$$

Overlined variables denote sequences of F *and* T*, where* $lvl(\bar{x}, z)$ *says that sequence* $\bar{x}$ *has length* 2^z*. Initial sequences have length* 1 (2)*. Longer sequences emerge from concatenations* (3) *with equal-length parts* (4)*. The tgd* (5) *promotes concatenations* x *to sequences* $\bar{w}$ *on the next level* z_+*. We mark such new sequences separately* (6)*, since this will later be useful. The tgds* (2)–(6) *on* $\mathcal{D}_{lvl}$ *therefore construct all* F*-*T*-sequences of length* 2^ℓ*, i.e.,* 2^{2^ℓ} *distinct nulls.*

Then $\Omega_v = \{\langle cat, 4\rangle, \langle part, 2\rangle, \langle up, 1\rangle\}$ *and* $\Omega_{\bar{w}} = \{\langle up, 3\rangle, \langle lvl, 1\rangle, \langle cat, 1\rangle, \langle cat, 2\rangle\}$*. The three edges of* $\mathsf{LXG}(\Sigma)$ *are* $v \xrightarrow{x(5)} \bar{w}$*,* $\bar{w} \xrightarrow{\bar{x}_1(3)} v$*, and* $\bar{w} \xrightarrow{\bar{x}_2(3)} v$*, where we disambiguate some variables with the numbers of their tgds.*

Definition 2 slightly sharpens the original definition by introducing edges only based on frontier variables. Moreover, by adding labels, a single edge in the original graph now corresponds to one or more edges in $\mathsf{LXG}(\Sigma)$. Therefore, if the existential dependency graph contains no cycles (i.e., Σ is jointly acyclic), then the labelled existential dependency graph does not contain cycles either.

$\mathsf{LXG}(\Sigma)$ is useful since it over-estimates the possible data flow in the computation of $\mathsf{chase}(\Sigma, \mathcal{D})$. To make this precise, note that any null n in $\mathsf{chase}(\Sigma, \mathcal{D})$ is introduced in a chase step i as a fresh value $n = \sigma^+[i](v)$ of some existential variable v in $\mathsf{tgd}[i]$. We write $\mathsf{var}(n)$ for this v, and $t \overset{y}{\twoheadrightarrow} n$ to indicate that $\sigma[i](y) = t$ for some term $t \in \mathbf{C} \cup \mathbf{N}$ and frontier variable y of $\mathsf{tgd}[i]$.

Lemma 4. *If* $n_1 \overset{y}{\twoheadrightarrow} n_2$ *in* $\mathsf{chase}(\Sigma, \mathcal{D})$ *for nulls* $n_1, n_2 \in \mathbf{N}$*, then there is an edge* $\mathsf{var}(n_1) \xrightarrow{y} \mathsf{var}(n_2)$ *in* $\mathsf{LXG}(\Sigma)$*.*

Lemma 5. *If* $\mathsf{chase}(\Sigma, \mathcal{D})$ *is infinite, then* $\mathsf{chase}(\Sigma, \mathcal{D})$ *contains an infinite chain* $n_0 \overset{y_1}{\twoheadrightarrow} n_1 \overset{y_2}{\twoheadrightarrow} \cdots$*.*

By Lemma 4, the infinite path of Lemma 5 corresponds to an infinite path in $\mathsf{LXG}(\Sigma)$, which must therefore, since it is finite, contain a cycle. Hence, if $\mathsf{LXG}(\Sigma)$ is acyclic, $\mathsf{chase}(\Sigma, \mathcal{D})$ is finite.

Example 6. *For* Σ *as in Example 3,* $\mathsf{LXG}(\Sigma)$ *is cyclic. Indeed, there are databases* $\mathcal{D}$ *for which* $\mathsf{chase}(\Sigma, \mathcal{D})$ *is infinite, e.g., for* $\mathcal{D} = \{first(1), last(1), next(1, 1)\}$*.*

4 TERMINATION WITH CYCLES

In this section, we establish decidable criteria to show that the chase on a tgd set is guaranteed to be finite, for all input databases, even though the existential dependency graph has some cycles.

Whenever a tgd $B[\boldsymbol{x}, \boldsymbol{y}] \to \exists \boldsymbol{v}.H[\boldsymbol{y}, \boldsymbol{v}]$ was applied in step i of $\mathsf{chase}(\Sigma, \mathcal{D})$, the conjunctive query $(B \wedge H)[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{v}]$ has the answer $\sigma^+[i]$ over $\mathsf{chase}(\Sigma, \mathcal{D})$. Likewise, a chain of chase steps (or path in $\mathsf{LXG}(\Sigma)$) leads to a match for a larger query, defined next.

Definition 7. *Consider a path* $\mathfrak{p} = v_0 \xrightarrow{y_1} v_1 \xrightarrow{y_2} \cdots \xrightarrow{y_k} v_k$ *in* $\mathsf{LXG}(\Sigma)$ *for variables* $v_0, v_i, y_i \in \mathbf{V}$ *(*$1 \le i \le k$*). Let* $\rho_i : B_i[\boldsymbol{x_i}, \boldsymbol{y_i}] \to \exists \boldsymbol{v_i}.H_i[\boldsymbol{y_i}, \boldsymbol{v_i}]$ *be a variant of the tgd* ρ_{v_i} *of variable* v_i*, where variables have been bijectively renamed such that tgds for different steps do not share variables. For* $1 \le i \le k$*, let* $\tilde{y}_i$ *(and* $\tilde{v}_i$*) denote the renamed version of* y_i *(and* v_i*), and let* $\tilde{y}_{k+1}$ *denote a fresh variable.*

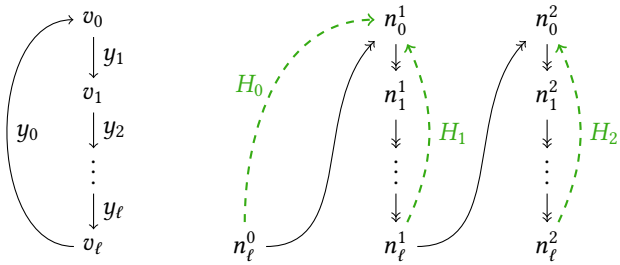


Fig. 1. Dependency cycle in LXG(Σ) (left) and corresponding chain of derived nulls in chase(Σ, $\mathcal{D}$) (right); H_0, H_1, and H_2 denote variants of a tgd head to illustrate propagation

We define $\mathrm{Path}(\mathfrak{p}) := \bigwedge_{i=1}^{k}(B_i \wedge H_i[\tilde{v}_i/\tilde{y}_{i+1}])$. *Given* $1 \le i \le k$, *we write* $\mathrm{Path}(\mathfrak{p})_{B,i} := B_i$ *and* $\mathrm{Path}(\mathfrak{p})_{H,i} := H_i[\tilde{v}_i/\tilde{y}_{i+1}]$ *for the i-th body and head conjunction, respectively, with the renamed variables.*

Example 8. LXG(Σ) *of Example 3 has a path* $\mathfrak{p} = v \xrightarrow{x} \bar{w} \xrightarrow{\bar{x}_1} v$, *where we omit the numbers of the tgds from variables for simplicity. Then* $\mathrm{Path}(\mathfrak{p}) = cat(\bar{x}_1', \bar{x}_2', z', x) \wedge next(z', z_+') \wedge up(x, z_+', \bar{x}_1) \wedge lvl(\bar{x}_1, z'') \wedge lvl(\bar{x}_2'', z'') \wedge cat(\bar{x}_1, \bar{x}_2'', z'', y_3)$. *Variables marked with ′ and ″ stem from renamed variants of tgds* (3) *and* (5), *respectively. Similarly,* $\mathrm{Path}(\mathfrak{p})_{H,1} = up(x, z_+', \bar{x}_1)$.

Lemma 9. *Let* $\mathfrak{c} = n_0 \overset{y_1}{\twoheadrightarrow} n_1 \overset{y_2}{\twoheadrightarrow} \cdots \overset{y_k}{\twoheadrightarrow} n_k$ *be a chain in* chase(Σ, $\mathcal{D}$), *with* n_k *derived in the m-th chase step* $\mathcal{D}^m$. *There is a path* $\mathfrak{p} = \mathrm{var}(v_0) \xrightarrow{y_1} \mathrm{var}(n_1) \xrightarrow{y_2} \cdots \xrightarrow{y_k} \mathrm{var}(n_k)$ *in* LXG(Σ), *and* $\mathcal{D}^m \models \mathrm{Path}(\mathfrak{p})\theta$ *holds for substitution* θ *defined as follows: for each* n_i *derived in chase step j, and each variable x in* $\mathrm{tgd}[j]$ *that was renamed to* $\tilde{x}$ *in* $\mathrm{Path}(\mathfrak{c})$, $\theta(\tilde{x}) = \sigma^+[j](x)$.

Lemma 9 ensures that every chain of nulls in the chase is accompanied by facts of the form Path(𝔭), connecting all nulls of the chain. Figure 1 sketches this situation for a cyclic path (left). A corresponding chain of nulls $n_\ell^0 \overset{y_0}{\twoheadrightarrow} n_0^1 \overset{y_1}{\twoheadrightarrow} \ldots$ is sketched on the right, where vertical positions indicate existential variables, i.e., $\mathrm{var}(n_i^j) = v_i$. Moreover, a solid arrow like $n_\ell^0 \twoheadrightarrow n_0^1$ also corresponds to facts of the form $\mathrm{Path}(n_\ell^0 \overset{y_0}{\twoheadrightarrow} n_0^1)$ that connect the respective nulls. The dashed arrows are explained below.

We can use the facts of Path(𝔭) to infer additional information that may help with chase termination for cyclic paths. Indeed, additional information may prevent tgd applications if the head of a tgd is already entailed (Definition 1 (2.a)).

To understand this better, let's denote the tgd for edge $v_\ell \xrightarrow{y_0} v_0$ in Figure 1 as $\rho = B[\boldsymbol{x}, \boldsymbol{y}] \to \exists \boldsymbol{v}.H[\boldsymbol{y}, \boldsymbol{v}]$ with $y_0 \in \boldsymbol{y}$ and $v_0 \in \boldsymbol{v}$. If i is the chase step that introduced n_0^1, then $H\sigma^+[i] \subseteq \mathcal{D}^{i+1}$. In Figure 1, H_0 represents the facts $H\sigma^+[i]$, which involve (among others) the terms n_ℓ^0 and n_0^1, as suggested by the dashed arrow.

Clearly, we cannot apply ρ twice for the same substitution of its frontier: for all $i' > i$ and substitutions σ with $\boldsymbol{y}\sigma = \boldsymbol{y}\sigma[i]$, we have $\mathcal{D}^{i'} \models \exists \boldsymbol{v}.H\sigma$ due to the presence of H_0. However, a cycle in LXG(Σ) can lead to a chain of tgd applications as in Figure 1, where ρ is applied to different frontiers in several steps. Indeed, if j is the chase step that introduced n_0^2, then $y_0\sigma[i] \neq y_0\sigma[j]$, so the matches differ on at least this frontier variable. If we write $\boldsymbol{y}^- := \boldsymbol{y} \setminus \{y_0\}$ for the frontier of ρ without y_0, we can think of $\boldsymbol{y}^-\sigma[i]$ as the "context" for which ρ was applied to n_ℓ^0 in step i.

Our goal is that ρ can never be applied twice to the same context within a single chain. To ensure this, we would like the chase to derive additional facts H_1 and H_2 as indicated in

Figure 1. Fixing a variable order $H[y_0, \boldsymbol{y^-}, \boldsymbol{v}]$, these facts are $H_1 := H[n^1_\ell, \boldsymbol{y^-}\sigma[i], \boldsymbol{v}\sigma^+[i]]$ and $H_2 := H[n^2_\ell, \boldsymbol{y^-}\sigma[i], \boldsymbol{v}\sigma^+[j]]$. Chains like those in Figure 1, even if finite, can become rather long, and we need facts H_p to be propagated to all nulls n^p_ℓ. We therefore require two kinds of conditions: (i) a base case that turns recently derived (forward) H_0 into a (backwards) H_1, and (ii) an inductive step that propagates a (backwards) H_p to another (backwards) H_{p+1}. The next definition spells out the two conditions. We generalise slightly by allowing that, instead of applying a single tgd ρ several times, the propagation can occur between different tgds along a path.

Definition 10. *For $p \in \{*, a, b\}$, let $u^p \xrightarrow{y^p} v^p \in \mathsf{LXG}(\Sigma)$ be such that the corresponding tgd ρ^p has a head $\exists v^p, \boldsymbol{w^p}.H^p[y^p, \boldsymbol{z^p}, v^p, \boldsymbol{w^p}]$. Moreover, let Σ_{DL} denote the set of Datalog rules in Σ.*

A path $\mathfrak{p} = u^* \xrightarrow{y^*} v^* \xrightarrow{z_2} \ldots \xrightarrow{z_\ell} w$ is base-propagating *if*

$$\Sigma_{\mathsf{DL}} \models \mathrm{Path}(\mathfrak{p}) \rightarrow H^*[\tilde{y}_{\ell+1}, \tilde{\boldsymbol{z}}_{\boldsymbol{1}}, \tilde{y}_2, \tilde{\boldsymbol{w}}_{\boldsymbol{1}}] \tag{7}$$

where the conclusion is the path's first head conjunction $\mathrm{Path}(\mathfrak{p})_{H,1} = H^*[\tilde{y}_1, \tilde{\boldsymbol{z}}_{\boldsymbol{1}}, \tilde{y}_2, \tilde{\boldsymbol{w}}_{\boldsymbol{1}}]$ with $\tilde{y}_1$ replaced by the variable $\tilde{y}_{\ell+1}$ that occurs in $\mathrm{Path}(\mathfrak{p})_{H,\ell} = H_\ell[\tilde{y}_\ell, \tilde{\boldsymbol{z}}_{\boldsymbol{\ell}}, \tilde{y}_{\ell+1}, \tilde{\boldsymbol{w}}_{\boldsymbol{\ell}}]$.

A composite path $\mathfrak{p}$ of the form $\mathfrak{p}^a\mathfrak{p}^b$ with

$$\mathfrak{p}^a = u^a \xrightarrow{y^a} v^a \xrightarrow{z_2} \ldots \xrightarrow{z_\ell} u^b \text{ and } \mathfrak{p}^b = u^b \xrightarrow{y^b} v^b \xrightarrow{z'_2} \ldots \xrightarrow{z'_k} w$$

is step-propagating *for ρ^* (or, equivalently, for $u^* \xrightarrow{y^*} v^*$) if*

$$\Sigma_{\mathsf{DL}} \models \mathrm{Path}(\mathfrak{p}) \wedge H^*[\tilde{y}_{\ell+1}, \boldsymbol{x_z}, \tilde{y}_2, \boldsymbol{x_w}] \rightarrow H^*[\tilde{y}_{\ell+k+1}, \boldsymbol{x_z}, \tilde{y}_{\ell+2}, \boldsymbol{x_w}] \tag{8}$$

where $\boldsymbol{x_z}$ and $\boldsymbol{x_w}$ are lists of fresh variables of length $|\boldsymbol{x_z}| = |\boldsymbol{z^*}|$ and $|\boldsymbol{x_w}| = |\boldsymbol{w^*}|$, respectively; $\tilde{y}_{\ell+k+1}$ is the renamed version of the final variable w in $\mathrm{Path}(\mathfrak{p})$; and the other mentioned variables stem from the atom sets $\mathrm{Path}(\mathfrak{p})_{H,1} = H^a[\tilde{y}_1, \tilde{\boldsymbol{z}}_{\boldsymbol{1}}, \tilde{y}_2, \tilde{\boldsymbol{w}}_{\boldsymbol{1}}]$ and $\mathrm{Path}(\mathfrak{p})_{H,\ell+1} = H^b[\tilde{y}_{\ell+1}, \tilde{\boldsymbol{z}}_{\boldsymbol{\ell+1}}, \tilde{y}_{\ell+2}, \tilde{\boldsymbol{w}}_{\boldsymbol{\ell+1}}]$.

Intuitively speaking, considering Figure 1, base propagation allows us to infer H_1 from H_0, and step propagation allows us to infer H_2 from H_1. In contrast to the simplified situation in the figure, the definition clarifies that the propagated head H^* is not necessarily the head of the tgds ρ^a and ρ^b that occur in the current piece of chain we consider.

Example 11. *We extend Σ from Example 3 by two additional tgds:*

$$up(y_1, z, \bar{y}_2) \wedge part(\bar{y}_2, y_3) \rightarrow up(y_3, z, \bar{y}_2) \tag{9}$$

$$up(y_3, z, \bar{y}_2) \wedge part(\bar{y}_2, y_3) \wedge up(y_3, z', \bar{y}_4) \wedge part(\bar{y}_4, y_5) \rightarrow up(y_5, z, \bar{y}_4) \tag{10}$$

$\mathsf{LXG}(\Sigma)$ remains as in Example 3. The path $\mathfrak{p} = v \xrightarrow{x} \bar{w} \xrightarrow{\bar{x}_1} v$ from Example 8 is base-propagating. Indeed, tgds (4) *and* (9) *entail the required tgd $\mathrm{Path}(\mathfrak{p}) \rightarrow up(y_3, z'_+, \bar{x}_1)$, in fact even the stronger tgd $up(x, z'_+, \bar{x}_1) \wedge cat(\bar{x}_1, \bar{x}''_2, z'', y_3) \rightarrow up(y_3, z'_+, \bar{x}_1)$. Similarly, for $\mathfrak{p}^a = v \xrightarrow{x} \bar{w} \xrightarrow{\bar{x}_1} v$ and $\mathfrak{p}^b = v \xrightarrow{x} \bar{w} \xrightarrow{\bar{x}_2} v$, the path $\mathfrak{p}^a\mathfrak{p}^b$ is step-propagating for* (5) *(i.e., for $v \xrightarrow{x} \bar{w}$) using tgds* (4) *and* (10). *Note that variables y_i in* (10) *match the path labels in Definition 10.*

As we will see below, the extended example achieves universal termination, and in particular also terminates for the database of Example 6. For the case that next is a strict order, tgd (6) *ensures that each sequence is assigned a unique level, even with the additional up-facts from tgds* (9) *and* (10).

The complexity of checking Definition 10 is dominated by the ExpTime complexity of Datalog.

Lemma 12. *Checking whether a path $\mathfrak{p}$ is base-propagating (or step-propagating) is* ExpTime*-complete, and* P*-complete with respect to the length of $\mathfrak{p}$.*

The conditions of Definition 10 have the desired impact on the chase: if we find a sequence of nulls that corresponds (by Lemma 4) to a path that is propagating, then satisfied head atoms in the chase are propagated accordingly. Moreover, since propagation is defined based on Σ_{DL}, the Datalog-first chase ensures that the propagation happens before further nulls are introduced. To ensure termination, we require that the cycles in each strongly connected component in $\mathsf{LXG}(\Sigma)$ can be broken by removing a set of edges E that are connected by propagating paths:

Definition 13. *Let C be a strongly connected component in $\mathsf{LXG}(\Sigma)$, and let E be a set of edges in C. An $\bar{E}$-*path *is a path in C that (i) contains no edges from E, (ii) starts in some $s \in \{v \mid u \xrightarrow{y} v \in E\}$, and (iii) ends in some $t \in \{u \mid u \xrightarrow{y} v \in E\}$. Moreover, C is E-*saturating *if*

(1) C without the edges of E is acyclic;
(2) for all $u \xrightarrow{x} v, u' \xrightarrow{x'} v \in E$, we have $x = x'$;
(3) all paths $e\mathfrak{p}$ in C, such that $e \in E$ and $\mathfrak{p}$ is an $\bar{E}$-path, are base-propagating; and
(4) all paths $e^a\mathfrak{p}_1 e^b\mathfrak{p}_2$ in C, such that $e^a, e^b \in E$ and $\mathfrak{p}_1, \mathfrak{p}_2$ are $\bar{E}$-paths, are step-propagating for every $e \in E$.

Σ is saturating *if all strongly connected components in $\mathsf{LXG}(\Sigma)$ are E-saturating for some E.*

Example 14. *The extended set of tgds from Example 11 is saturating. $\mathsf{LXG}(\Sigma)$ has a single strongly connected component (cf. Example 3) where Definition 13 is satisfied for $E = \{v \xrightarrow{x} \bar{w}\}$. For $e = v \xrightarrow{x} \bar{w}$,* (3) *applies to two paths $e\mathfrak{p}$ and* (4) *to four paths $e\mathfrak{p}e\mathfrak{p}'$, where $\mathfrak{p}, \mathfrak{p}' \in \{\bar{w} \xrightarrow{\bar{x}_i} v \mid i \in \{1, 2\}\}$. Propagation follows as in Example 11.*

Theorem 15. *Deciding if C is E-saturating is* ExpTime*-complete in the size of C. The same complexity applies to deciding if a tgd set Σ is saturating.*

In practice, the exponential factors in Theorem 15 might be small, already because strongly connected components are often small. The next result states that tgds in the edge sets E can only be applied at most once for each substitution of their "context" variables – an essential ingredient for termination.

Lemma 16. *Let C be an E-saturating strongly connected component in $\mathsf{LXG}(\Sigma)$, and let $u \xrightarrow{y} v \in E$ have the corresponding tgd ρ_v with head $\exists v, \boldsymbol{w}.H[y, \boldsymbol{z}, v, \boldsymbol{w}]$. For every database $\mathcal{D}$ and chain of nulls $n_0 \overset{y}{\twoheadrightarrow} n_1 \overset{x_2}{\twoheadrightarrow} \dots \overset{x_{\ell-1}}{\twoheadrightarrow} n_{\ell-1} \overset{y}{\twoheadrightarrow} n_\ell$ in $\mathsf{chase}(\Sigma, \mathcal{D})$ such that $n_0 = y\sigma[a]$, $n_1 = v\sigma^+[a]$, $n_{\ell-1} = y\sigma[b]$, and $n_\ell = v\sigma^+[b]$ for chase steps $a < b$, we have $z\sigma[a] \neq z\sigma[b]$.*

The main result of this section is as follows. A more detailed analysis follows in the next section.

Theorem 17. *If Σ is saturating, then $\mathsf{chase}(\Sigma, \mathcal{D})$ is finite for all databases $\mathcal{D}$.*

Example 18. *Theorem 17 also subsumes previous termination results by Carral et al. [14]. The following tgds captures the essence of their approach of simulating finite sets in tgds:*

$$elem(x) \wedge set(S) \rightarrow \exists v.\, set(S) \wedge su(x, S, v) \wedge su(x, v, v) \tag{11}$$

$$su(x, S, T) \wedge su(y, S, S) \rightarrow su(y, T, T) \tag{12}$$

The database can provide elem-facts that define a domain of elements, and a fact $set(\emptyset)$. The tgd (11) *constructs new "sets" by creating facts $su(x, S, T)$, which can be read as $\{x\} \cup S = T$. In particular, $su(x, S, S)$ means $x \in S$. The tgd* (12) *propagates memberships $x \in S$ from S to a direct superset T, which we recognise as a special case of step propagation. Indeed, the only dependency here is $v \xrightarrow{S} v$, and all paths $\mathfrak{p}_{(1/2)}$ in Definition 13 are empty. Base propagation is achieved by the atom $su(x, v, v)$ in* (11) *without requiring any Datalog rules.*

Example 19. *Definition 13* (2) *excludes "confluent" edges with distinct labels: being based on single frontier variables, our propagation conditions do not suffice for this case. For example, consider the tgd set* Σ *that extend the tgd set of Example 18:*

$$set(S) \rightarrow elem(S) \tag{13}$$
$$su(x,S,T) \rightarrow su(T,S,T) \tag{14}$$
$$su(x,S,T) \wedge su(y,x,x) \rightarrow su(y,T,T) \tag{15}$$
$$su(x,S,T) \wedge su(S,y,S) \rightarrow su(T,y,T) \tag{16}$$
$$su(x,S,T) \wedge su(x,y,x) \rightarrow su(T,y,T) \tag{17}$$

We allow "sets" to be used as "elements" (13)*, thereby creating a confluent E-edge. The tgd* (14) *ensures that the new E-edge is base-propagating, and tgds* (15) – (17) *ensure that the additional pairs of E-edges are step-propagating. For* $\mathcal{D} = \{set(\emptyset)\}$, $\mathsf{chase}(\Sigma, \mathcal{D})$ *is infinite.*

However, strongly connected components may have arbitrary confluent edges that are not in E, as in Example 14, and E itself may have confluent edges that are using the same label.

5 COMPLEXITY OF THE SATURATING CHASE

Next, we refine Theorem 17 by deriving specific bounds for the size of the chase over saturating tgd sets Σ, based on the structure of $\mathsf{LXG}(\Sigma)$. For a vertex v of $\mathsf{LXG}(\Sigma)$, we write $\mathsf{SCC}(v)$ for the strongly connected component that contains v, and $\mathsf{SCC}(\mathsf{LXG}(\Sigma))$ for the set of all strongly connected components. An edge $u \xrightarrow{y} w$ is *incoming* for $\mathsf{SCC}(v)$ if $u \notin \mathsf{SCC}(v)$ and $w \in \mathsf{SCC}(v)$; in this case we write $\mathsf{SCC}(u) \prec \mathsf{SCC}(w)$ (where $\mathsf{SCC}(w) = \mathsf{SCC}(v)$). The transitive reflexive closure of $\prec$ is the usual induced partial order on $\mathsf{SCC}(\mathsf{LXG}(\Sigma))$.

Definition 20. *For vertex* v *of* $\mathsf{LXG}(\Sigma)$*, we define the label set* $\lambda(v) = \{x \mid u \in \mathsf{SCC}(v), u \xrightarrow{x} v \text{ in } \mathsf{LXG}(\Sigma)\}$*, and* confluence $\mathsf{conf}(v) = |\lambda(v)|$*. For* $C \in \mathsf{SCC}(\mathsf{LXG}(\Sigma))$*, we define* $\mathsf{conf}(C) = \max\{\mathsf{conf}(u) \mid u \in C\}$*.* C *is* homogeneously confluent *if* $\mathsf{conf}(C) = 1$*.*

Note that $\mathsf{conf}(C) = 0$ implies that C is trivial, i.e., a singleton set without any cycle (self loop).

Definition 21. *Consider a database* $\mathcal{D}$ *and* $C \in \mathsf{SCC}(\mathsf{LXG}(\Sigma))$*. A term* t *in* $\mathsf{chase}(\Sigma, \mathcal{D})$ *is a* C-input *if (i)* t *is a constant, or (ii)* t *is a null and* C *has an incoming edge* $\mathsf{var}(t) \xrightarrow{x} w$*. A null* n *with* $\mathsf{var}(n) \in C$ *has* C-depth k *if there is a maximal chain of nulls* $m_0 \overset{y_1}{\twoheadrightarrow} \ldots \overset{y_k}{\twoheadrightarrow} m_k$ *in* $\mathsf{chase}(\Sigma, \mathcal{D})$ *with* $m_k = n$ *and* $\mathsf{var}(m_i) \in C$ *for* $0 \leq i \leq k$*. The* C*-depth of* n *is undefined if the length of such chains is unbounded.*

The next result limits the number of bounded-C-depth nulls based on the number of C-inputs, thereby also clarifying the significance of homogeneous confluence. Note that this general insight does not restrict to saturating tgd sets.

Lemma 22. *Consider a database* $\mathcal{D}$ *and* $C \in \mathsf{SCC}(\mathsf{LXG}(\Sigma))$*. If* i *is the number of* C*-inputs in* $\mathsf{chase}(\Sigma, \mathcal{D})$*, then, for any* $d \geq 0$*, the number of nulls at* C*-depth* $\leq d$ *is at most doubly exponential in* d *and polynomial in* i*. If* C *is homogeneously confluent, then this number is at most exponential in* d*.*

Definition 23. *Let* Σ *be saturating with* E_C *denoting the set E of Definition 13 for* $C \in \mathsf{SCC}(\mathsf{LXG}(\Sigma))$*. Let* $C_1, \ldots, C_k$ *be a list of all* $C_i \in \mathsf{SCC}(\mathsf{LXG}(\Sigma))$ *topologically ordered w.r.t. the induced order* $\prec$*.*

We define $\mathsf{rank}(C_i)$ *iteratively for* $i = 1, \ldots, k$ *as follows, where we assume* $\max\{\} = 0$*. First, let* $r^i_{\mathsf{in}} := \max\{\mathsf{rank}(C_j) \mid C_j \prec C_i\}$ *and let* $r^i_{\mathsf{cxt}} := \max\{\mathsf{rank}(C) \mid C \in C^i_{\mathsf{cxt}}\}$ *where* C^i_{cxt} *denotes the set*

$$\{C_j \mid v, w \in C_i, u \in C_j, w \xrightarrow{x} v \in E_{C_i}, u \xrightarrow{y} v \in \mathsf{LXG}(\Sigma), x \neq y\}.$$

Then the rank of C_i is

$$\mathsf{rank}(C_i) := \begin{cases} r^i_{\mathsf{in}} & \text{if } \mathsf{conf}(C_i) = 0, \\ \max\{r^i_{\mathsf{in}}, r^i_{\mathsf{cxt}} + 1\} & \text{if } \mathsf{conf}(C_i) = 1, \\ \max\{r^i_{\mathsf{in}}, r^i_{\mathsf{cxt}} + 2\} & \text{if } \mathsf{conf}(C_i) \geq 2. \end{cases}$$

The rank of Σ is $\mathsf{rank}(\Sigma) = \max\{\mathsf{rank}(C) \mid C \in \mathsf{SCC}(\mathsf{LXG}(\Sigma))\}$.

Theorem 24. *Let Σ be saturating and $\mathcal{D}$ a database. For every existential variable v in Σ, the number of nulls n with $\mathsf{var}(n) = v$ in $\mathsf{chase}(\Sigma, \mathcal{D})$ is at most $\mathsf{rank}(\mathsf{SCC}(v))$-exponential in the size of $\mathcal{D}$.*

Example 14 (and the discussion in Example 3) shows that the upper bound of Theorem 24 can be reached. We can further strengthen this into a hardness result:

Theorem 25. *Let Σ be saturating. For every database $\mathcal{D}$ the size of $\mathsf{chase}(\Sigma, \mathcal{D})$ is at most $\mathsf{rank}(\Sigma)$-exponential in the size of $\mathcal{D}$, and BCQ entailment is $\mathsf{rank}(\Sigma)$-ExpTime-complete for data complexity.*

6 THE CHASE IN THE FOREST

In this section, we refine Theorem 25 by identifying cases where BCQ answering over saturating tgd sets Σ is not $\mathsf{rank}(\Sigma)$-ExpTime-complete but merely $(\mathsf{rank}(\Sigma) - 1)$-ExpSpace-complete, and we design a chase procedure that runs within these complexity bounds. To simplify presentation, we consider tgd sets with a single rank-maximal SCC $\hat{C}$ in $\mathsf{LXG}(\Sigma)$ (generalisations are possible; see concluding remarks). If $\mathsf{conf}(\hat{C}) = 1$, we can establish a tree-like search space within the chase that follows the edges of $\mathsf{LXG}(\Sigma)$. A new syntactic restriction on tgds, called *path guardedness*, ensures that a chase that follows this tree-like structure remains complete for conjunctive query answering.

Definition 26. *A tgd set Σ is* arboreous *if it is saturating, and has a unique $\hat{C} \in \mathsf{SCC}(\Sigma)$ with $\mathsf{rank}(\hat{C}) = \mathsf{rank}(\Sigma)$, which satisfies $\mathsf{conf}(\hat{C}) \leq 1$. For such Σ and some $\mathsf{chase}(\Sigma, \mathcal{D})$, the* null forest *is the directed graph $\langle \hat{N}, \hat{\twoheadrightarrow} \rangle$ with $\hat{N} = \{n \in \mathbf{N}(\mathsf{chase}(\Sigma, \mathcal{D})) \mid \mathsf{var}(n) \in \hat{C}\}$ the nulls of variables in $\hat{C}$, and $n \mathrel{\hat{\twoheadrightarrow}} m$ if $n \overset{x}{\twoheadrightarrow} m$ for some x.*

Lemma 27. *For every arboreous Σ and $\mathsf{chase}(\Sigma, \mathcal{D})$, the null forest is indeed a forest (set of trees).*

Next, we use the special tgds that correspond to set E of Definition 13 to partition the null forest into sub-forests. The intuition is that special tgd applications start a new sub-forest (their fresh nulls being the roots), whereas other tgd applications remain within their current sub-forest. By placing all remaining terms of the chase in an additional root node, we obtain a tree structure whose nodes are sets of terms that partition the terms of the chase:

Definition 28. *Let Σ, $\hat{C}$, $\hat{N}$, and $\hat{\twoheadrightarrow}$ be as in Definition 26, let $\hat{E}$ be the edge set E of Definition 13 for $\hat{C}$, and let T be the set of all terms in $\mathsf{chase}(\Sigma, \mathcal{D})$. An $\hat{E}$-*tgd *is a tgd with an existential variable that is the target of an edge in $\hat{E}$. The $\hat{E}$-*variables *$V_{\hat{E}}$ are all existential variables in $\hat{C}$ that occur in some $\hat{E}$-tgd.*

For every chase step i where an $\hat{E}$-tgd was applied, let $R[i]$ be the set of fresh nulls introduced in step i. Let $\bar{R} \subseteq \hat{N}$ be the set of nulls that are not in any such $R[i]$. Then, for each $R[i]$, let $F[i] \subseteq \hat{N}$ be the least set that contains $R[i]$ and all $m \in \bar{R}$ for which there is $n \in F[i]$ with $n \mathrel{\hat{\twoheadrightarrow}} m$. Let $\mathcal{F}$ be the set of all such sets $F[i]$. For $F_1, F_2 \in \mathcal{F}$ with $F_1 \neq F_2$, we write $F_1 \mathrel{\tilde{\twoheadrightarrow}} F_2$ if there are $n_i \in F_i$ $(i = 1, 2)$ such that $n_1 \mathrel{\hat{\twoheadrightarrow}} n_2$.

Let $\mathsf{L}_0 = T \setminus \bigcup_{F \in \mathcal{F}} F$. The term tree of $\mathsf{chase}(\Sigma, \mathcal{D})$ *is the graph $\langle N_\sim, \tilde{\twoheadrightarrow} \rangle$ with $N_\sim = \mathcal{F} \cup \{\mathsf{L}_0\}$, and $\tilde{\twoheadrightarrow}$ extended to $N_\sim$ by setting $\mathsf{L}_0 \mathrel{\tilde{\twoheadrightarrow}} F$ for all $F \in \mathcal{F}$ that have no $\tilde{\twoheadrightarrow}$-predecessor in $\mathcal{F}$. The reflexive transitive closure of $\tilde{\twoheadrightarrow}$ is denoted $\tilde{\twoheadrightarrow}^*$.*

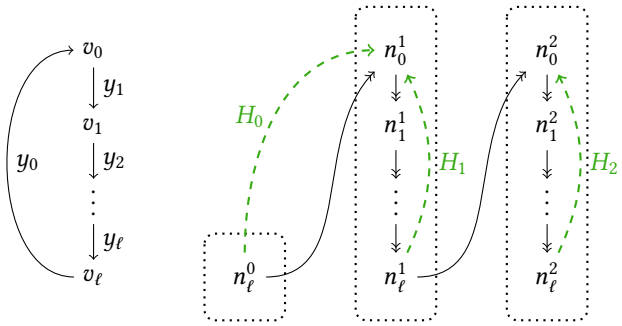


Fig. 2. Example graph LXG(Σ) (left) and chain of derived nulls in chase(Σ, $\mathcal{D}$) (right) from Figure 1, with nulls partitioned according to Definition 28 for the set $\hat{E} = \{v_\ell \xrightarrow{y_0} v_0\}$

Lemma 29. *For every arboreous* Σ *and* chase(Σ, $\mathcal{D}$), $N_\sim$ *is a partition of the terms in* chase(Σ, $\mathcal{D}$), *and the term tree is indeed a tree.*

For every term t, Lemma 29 allows us to use $\mathsf{L}(t)$ to denote the unique set $\mathsf{L} \in N_\sim$ with $t \in \mathsf{L}$.

Example 30. *Figure 2 revisits the abstract example from Figure 1, which we assume to be saturating according to Definition 13 using* $E = \{v_\ell \xrightarrow{y_0} v_0\}$. *If this is the unique maximal SCC* $\hat{C}$, *then* $\hat{E} = E$, *and we obtain three partitions of nulls that are illustrated in Figure 2:* $\mathsf{L}(n_\ell^0)$, $\mathsf{L}(n_0^1) = \mathsf{L}(n_1^1) = \cdots = \mathsf{L}(n_\ell^1)$, *and* $\mathsf{L}(n_0^2) = \mathsf{L}(n_1^2) = \cdots = \mathsf{L}(n_\ell^2)$. *These partitions are part of a path in the term tree, which is overlaid on the original null forest.*

The motivation for defining such a coarser tree structure is that we intend to use this tree to guide the computation of facts during the chase. We will limit its space-complexity by storing, at each particular moment during the chase, only those facts that can be represented using terms on a single path of this tree. The coarser the tree structure, the more facts can be considered at any moment, the more cases can be handled by this limited form of chase.

Besides this general intuition, the factorisation also plays a crucial role in deriving a syntactic criterion to recognise cases where such a tree-based chase can safely be applied, which we will consider next. The difficulty for this endeavour is that any such syntactic condition eventually has to rely on the facts that have induced the tree-like structure in the first place, such as H_0 *in Figure 2. But these very facts also occur in backwards direction to ensure saturation, as indicated by* H_1 *and* H_2 *in the figure. In the coarser tree structure, such "backward edges" merely lead to the same node of the tree, rather than to a predecessor node from which we could enter parallel branches in forward direction.*

The tree structure of terms as such does not constrain the structure of inferred facts in chase(Σ, $\mathcal{D}$), which may relate nulls from arbitrary positions in the null forest. We seek syntactic restrictions that ensure that the chase respects the term tree in the sense that the terms of any fact are on a common path and impose an order on terms that matches their position in the path. To this end, we derive relationships $\langle p, i\rangle \preceq \langle p, j\rangle$ on predicate positions such that, for all $p(\boldsymbol{t}) \in$ chase(Σ, $\mathcal{D}$) with $t_i, t_j \in \hat{N}$, we have that $\mathsf{L}(t_i)$ is an ancestor of (or possibly equal to) $\mathsf{L}(t_j)$ in the term tree. The next definition once again uses our assumption that distinct tgds do not share variables.

Definition 31. *Let* Σ *be arboreous with* $\hat{C}$, $\hat{E}$, *and* $V_{\hat{E}}$ *as in Definition 28. We will define a (not necessarily transitive) binary relation* $\preceq$ *on predicate positions* $\langle p, i\rangle$. *Any such* $\preceq$ *induces a relation* $\trianglelefteq$ *on variables of* Σ *as the reflexive, transitive closure of the set of all* $x \trianglelefteq y$ *such that* x *and* y *occur at positions* $\langle p, i\rangle$ *and* $\langle p, j\rangle$ *in a single body atom of some tgd in* Σ, *and* $\langle p, i\rangle \preceq \langle p, j\rangle$.

Algorithm 1: Non-deterministic chase for BCQ entailment

In: tgd set Σ, database $\mathcal{D}$, BCQ q, variable set $V_{\hat{E}}$, integer M
Out: $\Sigma \cup \mathcal{D} \models q$

$\mathcal{I} \coloneqq \mathcal{D}$
$\mathcal{T} \coloneqq \langle \mathbf{C}_0 \rangle$ with C_0 the set of all constants in Σ and $\mathcal{D}$
for $i \in \{1, \dots, |q|\}$ **do**
  for $j \in \{1, \dots, M\}$ **do**
    choose $\rho \in \Sigma$ with frontier $\boldsymbol{y}$, and match σ applicable to $\mathcal{I}$ in Datalog-first chase **or** **break** (go to (L3))
    while $|\mathcal{T}| > 1$ **and** $\boldsymbol{y}\sigma \cap \mathcal{T}.\mathtt{last}() = \emptyset$ **do**
      $\mathcal{T}.\mathsf{pop}()$
    if $\boldsymbol{v} \cap V_{\hat{E}} \neq \emptyset$ **then**
      $\mathcal{T}.\mathsf{push}(\boldsymbol{v}\sigma^+)$
    else
      $\mathcal{T}.\mathsf{push}(\mathcal{T}.\mathsf{pop}() \cup \boldsymbol{v}\sigma^+)$
    $\mathcal{I} \coloneqq \{p(\boldsymbol{t}) \in (\mathcal{I} \cup \mathsf{head}(\rho)\sigma^+) \mid \boldsymbol{t} \subseteq \bigcup_{T \in \mathcal{T}} T\}$
  $\mathcal{T} \coloneqq \langle \bigcup_{T \in \mathcal{T}} T \rangle$
return $\mathcal{I} \models q$

Now, concretely, let $\preceq$ denote the largest *relation on predicate positions such that, for all head atoms $p(\boldsymbol{t})$ in tgds of Σ with universally quantified variables $\boldsymbol{y}$ and existentially quantified variables $\boldsymbol{v}$:*

(1) if $t_i \in \boldsymbol{v} \cap \hat{C}$ and $t_j \in \boldsymbol{v} \setminus \hat{C}$ then $\langle p, i\rangle \not\preceq \langle p, j\rangle$,
(2) if $t_i \in \boldsymbol{v} \cap V_{\hat{E}}$ and $t_j \in \boldsymbol{y}$ then $\langle p, i\rangle \not\preceq \langle p, j\rangle$,
(3) if $t_i \in \boldsymbol{v} \cap \hat{C}$ and $t_j \in \boldsymbol{y}$ with $t_j \notin \lambda(t_i)$ then $\langle p, i\rangle \not\preceq \langle p, j\rangle$,
(4) if $t_i, t_j \in \boldsymbol{y}$ and $t_i \not\trianglelefteq t_j$ then $\langle p, i\rangle \not\preceq \langle p, j\rangle$.

One can construct $\preceq$ in polynomial time with a simple greatest fixed point computation. Note that such a construction is anti-monotonic in Σ: more tgds lead to fewer constraints $\preceq$.

Lemma 32. *Let Σ be arboreous. For every $\mathcal{D}$, if $p(t_1, \dots, t_n) \in \mathsf{chase}(\Sigma, \mathcal{D})$ with $\langle p, i\rangle \preceq \langle p, j\rangle$, then $\mathsf{L}(t_i) \tilde{\twoheadrightarrow}^* \mathsf{L}(t_j)$.*

Definition 33. *Let Σ be arboreous with $\hat{C}$ and $\trianglelefteq$ as in Definition 31. The set of $\hat{C}$-*affected *positions $\hat{\Omega}$ in Σ is $\bigcup_{v \in \hat{C}} \Omega_v$, with Ω_v as in Definition 2. A body variable x is $\hat{C}$-*affected *if $\mathrm{Pos}^B_x \subseteq \hat{\Omega}$.*

A tgd $\rho \in \Sigma$ is path-guarded *if all $\hat{C}$-affected body variables in ρ are mutually comparable with respect to the relation $\trianglelefteq$, i.e., form a chain in $\trianglelefteq$. Σ is* path-guarded *if all of its tgds are.*

Any node L of the term tree induces a unique upwards path $\mathsf{path}(\mathsf{L})$ that consists of all nodes L' with $\mathsf{L}' \tilde{\twoheadrightarrow}^* \mathsf{L}$. For term t, we write $\mathsf{path}(t)$ for $\mathsf{path}(\mathsf{L}(t))$. Inferences of path-guarded tgds are situated on such paths:

Lemma 34. *Let Σ be arboreous and path-guarded, and $\mathcal{D}$ some database. For every step i in $\mathsf{chase}(\Sigma, \mathcal{D})$, with $\mathsf{tgd}[i] = B[\boldsymbol{x}, \boldsymbol{y}] \to \exists \boldsymbol{v}.H[\boldsymbol{y}, \boldsymbol{v}]$, and $T_B = (\boldsymbol{x} \cup \boldsymbol{y})\sigma[i]$ and $T_H = (\boldsymbol{y} \cup \boldsymbol{v})\sigma^+[i]$:*

(1) if $T_B \neq \emptyset$ then $T_B \subseteq \bigcup \mathsf{path}(t)$ for some $t \in T_B$, and
(2) if $T_H \neq \emptyset$ then $T_H \subseteq \bigcup \mathsf{path}(t)$ for some $t \in T_H$.

Algorithm 1 specifies a non-deterministic chase procedure to check the entailment of a BCQ. It is intended for arboreous, path-guarded tgd sets Σ with variables $V_{\hat{E}}$ as in Definition 28. The input M

bounds the length of the search: we will determine a $\mathsf{rank}(\Sigma)$-exponential value for M for which the algorithm decides query entailment in $(\mathsf{rank}(\Sigma)-1)$-NExpSpace, showing the problem to be in $(\mathsf{rank}(\Sigma)-1)$-ExpSpace by Savitch's Theorem.

Algorithm 1 maintains a set of inferences $\mathcal{I}$ over terms in $\mathcal{T}$, which is a list of sets of terms that corresponds to a path in the term tree. We use operations `push`, `pop`, and `last`, respectively, to add, remove, or read $\mathcal{T}$'s last element. The algorithm performs a search for each atom in q (L3), adding one current path to $\mathcal{T}$'s root node after each run (L13). The inner loop (L4) non-deterministically chooses (L5) a tgd to apply to $\mathcal{I}$, or to break the iteration early (even if tgds are applicable). The current path $\mathcal{T}$ is pruned so that its last element contains a frontier term of the tgd (L6), before we either add a new term set (L9) or augment the last term set (L11), depending on whether ρ is an $\hat{E}$-tgd. Finally, we add the inferred head and restrict $\mathcal{I}$ to atoms with terms in $\mathcal{T}$ (L12), where σ^+ extends σ to existentially quantified variables using globally fresh nulls (not used in the algorithm before). Finally, we check if $\mathcal{I}$ entails q (L14).

Given a run of Algorithm 1, we write $\mathcal{I}_i^j$ (resp. $\mathcal{T}_i^j$) to denote the value of $\mathcal{I}$ (resp. $\mathcal{T}$) after executing (L12) in the ith iteration of loop (L3) and the jth iteration of (L4). Although Algorithm 1 can forget inferences and repeat the same tgd application with distinct fresh nulls, its computations are correct in the following sense:

Lemma 35. *Let $\mathcal{I}^*$ be the union of all sets $\mathcal{I}_i^j$ of some run of Algorithm 1. There is a homomorphism $\tau : \mathcal{I}^* \to \mathsf{chase}(\Sigma, \mathcal{D})$, and therefore $\mathsf{chase}(\Sigma, \mathcal{D}) \models q$ whenever Algorithm 1 returns* true.

Lemma 36. *If Σ is arboreous and path-guarded with $\mathsf{rank}(\Sigma) > 0$, and $M \leq f(|\mathcal{D}|)$ for some $\mathsf{rank}(\Sigma)$-exponential function f, then there is a $(\mathsf{rank}(\Sigma)-1)$-exponential function g such that Algorithm 1 runs in space $g(|\mathcal{D}|)$, where 0-exponential means polynomial.*

It remains to show that, whenever $\Sigma, \mathcal{D} \models q$, Algorithm 1 admits a run that returns true and is bounded by an M as in Lemma 36. Any run corresponds to a sequence of choices for (L5), which consists of $|q|$ sequences $\boldsymbol{s}_j$ of tgd applications. Let ρ and σ define the tgd application used to compute $\mathcal{I}_{i+1}^j$ from $\mathcal{I}_i^j$. We say that this tgd application *corresponds to chase step s* if $\mathsf{tgd}[s] = \rho$, the restriction of τ to terms in $\mathcal{T}_i^j$ is injective, and $\sigma(z) = \tau^-(\sigma[s](z))$. In this case, we *canonically extend* τ to the fresh nulls by $\tau(v\sigma^+) := v\sigma^+[s]$ for all existential variables v in ρ. This makes τ locally injective:

Lemma 37. *If all tgd applications in a run of Algorithm 1 correspond to chase steps, and τ in each iteration is the canonical extension for the respective step, then τ is a homomorphism $\mathcal{I}^* \to \mathsf{chase}(\Sigma, \mathcal{D})$ that is injective on all term sets $\bigcup \mathcal{T}_i^j$ that occur during the run.*

For completeness of the algorithm, we are interested in runs where all tgd applications correspond to chase steps. Such runs can be specified as a list of $|q|$ sequences of chase steps, where repetitions of steps are allowed (and sometimes necessary).

We obtain a suitable choice sequence by scheduling *tasks* $\langle d, i\rangle$, to be read as "perform chase step i under the assumption that $\mathcal{I}$ already contains (isomorphic copies of) all atoms that can be expressed using terms from the first $d-1$ elements of the path of Lemma 34 (1)." Such tasks may require other tasks to be completed first, since $\mathcal{I}$ may not yet contain the whole premise of $\mathsf{tgd}[i]$:

Definition 38. *For chase step i, let $\mathsf{path}_B(i)$ denote the path of Lemma 34 (1). For an atom $\alpha \in \mathsf{chase}(\Sigma, \mathcal{D})$, let $\mathsf{path}(\alpha)$ be the smallest path in the term tree that contains all terms of α (which are on a path by induction over Lemma 34 (2)).*

The subtasks *of a task $\langle d, i\rangle$ are all tasks $\langle e, j\rangle$ with $e \geq d$ a depth, and $j < i$ the largest chase step that produced an atom $\alpha \in \mathcal{D}^{j+1} \setminus \mathcal{D}^j$ with $|\mathsf{path}(\alpha)| = e$ and $\mathsf{path}(\alpha) \subseteq \mathsf{path}_B(i)$. The* task tree

for $\langle d, i\rangle$ has a root node with label $\langle d, i\rangle$ and the task trees for all subtasks of $\langle d, i\rangle$ as its children. Children of a single parent are ordered by the depth in their label: $\langle e, j\rangle < \langle e', j'\rangle$ if $e < e'$.

The atom α in Definition 38 ensures that the application of $\mathsf{tgd}[j]$ in Algorithm 1 will not delete any previous inferences up to depth e (through (L6) and (L12)). The order of subtasks ensures that inferences at smaller depths are computed first. Now the required sequence to successfully perform chase step i in the inner loop of Algorithm 1 is obtained by traversing the task tree for $\langle 1, i\rangle$ in a topological, order-respecting way (children before parents, smaller sibling nodes before larger ones), extracting the sequence of chase steps from the second component of the sequence of tasks.

Lemma 39. *If Σ is arboreous and path-guarded, $\boldsymbol{s}$ is the sequence of chase steps obtained from the task tree with root $\langle 1, i\rangle$, then the length $|\boldsymbol{s}|$ of $\boldsymbol{s}$ is bounded by a $\mathsf{rank}(\Sigma)$-exponential function.*

Lemma 40. *If Σ is arboreous and path-guarded, $\boldsymbol{s}$ is the sequence of chase steps obtained from the task tree for $\langle 1, i\rangle$, and $M \geq |\boldsymbol{s}|$, then Algorithm 1 can choose tgd applications according to $\boldsymbol{s}$.*

Combining these results, we obtain the completeness of our tree-based chase. Indeed, whenever $q\theta \subseteq \mathsf{chase}(\Sigma, \mathcal{D})$ for some match θ, there are chase steps $s_1, \ldots, s_{|q|}$ that produce the atoms of $q\theta$. A suitable strategy then executes Algorithm 1 for the choice sequences $\boldsymbol{s_i}$ obtained from the task trees for $\langle 1, s_i\rangle$ with $i = 1, \ldots, |q|$. Lemma 39 ensures that we can use a suitably small bound M to use the complexity results of Lemma 36, whereas Lemma 40 ensures that the final atom set $\mathcal{I}^{|q|}_{|\boldsymbol{s}_{|q|}|}$ contains all atoms of $\tau^-(q\theta)$.

Theorem 41. *BCQ entailment for arboreous and path-guarded tgd sets Σ with $\mathsf{rank}(\Sigma) = \kappa \geq 1$ is $(\kappa - 1)$-ExpSpace-complete for data complexity, where 0-ExpSpace = PSpace.*

Hardness is shown by reduction from the word problem of κ-exponentially time-bounded alternating Turing machines. PSpace-hardness is illustrated with a simpler reduction from TrueQBF:

Example 42. *Let $\phi = \mathsf{Q}_1 p_1, \ldots, \mathsf{Q}_\ell p_\ell.\, \psi$ be a quantified Boolean formula with $\mathsf{Q}_i \in \{\exists, \forall\}$ for $1 \leq i \leq \ell$ and $\psi = \bigwedge_{j=1}^{k} C^j$ in CNF. Let $\tilde{p}_i$ denote p_i or its negation $\bar{p}_i$; the clauses C^i are sets of such literals.*

The facts $\mathcal{D}_0 = \{\mathit{empty}(\emptyset), \mathit{new}(\vdash, \emptyset), \mathit{nxt}(\vdash, p_1), \mathit{nxt}(\vdash, \bar{p}_1)\} \cup \{\mathit{nxt}(\tilde{p}_i, \tilde{p}_{i+1}) \mid 1 \leq i < \ell\}$ encode an order over the literals. The tgds Σ_0 construct literal sets (truth assignments) similar to Example 18:

$$\mathit{getSU}(x, S) \to \exists v.\, \mathit{su}(x, S, v) \wedge \mathit{su}(x, v, v) \tag{18}$$

$$\mathit{su}(x, S, T) \wedge \mathit{su}(y, S, S) \to \mathit{su}(y, T, T) \tag{19}$$

$$\mathit{new}(x, S) \wedge \mathit{nxt}(x, y) \to \mathit{getSU}(y, S) \tag{20}$$

$$\mathit{new}(x, S) \wedge \mathit{nxt}(x, y) \wedge \mathit{su}(y, S, T) \to \mathit{new}(y, T) \tag{21}$$

Facts $\mathit{new}(l, S)$ mark the latest literal l added to a set S. The tgd (20) triggers the creation of a suitable set, and tgd (21) marks the newly added literal. Note that we never have $\mathit{su}(p_i, s, s)$ and $\mathit{su}(\bar{p}_i, s, s)$.

Now $\mathcal{D}_1 = \{\mathit{in}(\tilde{p}_i, C) \mid \tilde{p}_i \in C^j, 1 \leq j \leq k\} \cup \{\mathit{next}(C^{j-1}, C^j) \mid 1 < j \leq k\} \cup \{\mathit{first}(C^1), \mathit{last}(C^k)\}$ encodes the clauses. We use tgds Σ_1 to evaluate ψ under a given truth assignment:

$$\mathit{su}(x, S, S) \wedge \mathit{in}(x, c) \wedge \mathit{first}(c) \to \mathit{sat}(c, S) \tag{22}$$

$$\mathit{sat}(c, S) \wedge \mathit{next}(c, d) \wedge \mathit{su}(x, S, S) \wedge \mathit{in}(x, d) \to \mathit{sat}(d, S) \tag{23}$$

$$\mathit{sat}(c, S) \wedge \mathit{last}(c) \to \mathit{sat}(S) \tag{24}$$

Here, $\mathit{sat}(c, s)$ means "set s satisfies clause c". The tgds Σ_1 then propagate satisfaction along the order of $\mathcal{D}_1$. Therefore, $\mathsf{chase}(\Sigma_0 \cup \Sigma_1, \mathcal{D}_0 \cup \mathcal{D}_1) \models \exists v.\, \mathit{sat}(v)$ if and only if ψ is satisfiable.

Finally, we use $\mathcal{D}_2 = \{ex(p_i) \mid ?_i = \exists, 1 \leq i \leq \ell\} \cup \{pos(p_i), neg(\bar{p}_i) \mid 1 \leq i \leq \ell\}$ *and the following tgds* Σ_2 *to evaluate* ϕ*:*

$$su(x, S, T) \wedge ex(x) \wedge sat(T) \rightarrow sat(S) \tag{25}$$

$$su(x, S, T) \wedge pos(x) \wedge sat(T) \rightarrow sat_+(S) \tag{26}$$

$$su(x, S, T) \wedge neg(x) \wedge sat(T) \rightarrow sat_-(S) \tag{27}$$

$$sat_+(S) \wedge sat_-(S) \rightarrow sat(S) \tag{28}$$

We check satisfaction by evaluating the tree of literal sets by propagating satisfaction from leafs towards the root. The tgd (25) *handles existential quantification, and tgds* (26)–(28) *handle universal quantification. Handling the successors of universal states separately ensures that the tgds are path-guarded. Indeed,* $\preceq$ *of Definition 31 for the used tgds contains* $\langle su, 2\rangle \preceq \langle su, 3\rangle$*. Note that* $\mathsf{chase}(\Sigma_0 \cup \Sigma_1 \cup \Sigma_2, \mathcal{D}_0 \cup \mathcal{D}_1 \cup \mathcal{D}_2) \models \exists v.\, empty(v) \wedge sat(v)$ *if and only if* ϕ *is true.*

7 CONCLUSIONS AND OUTLOOK

We have established new criteria for chase termination, which advance the state of the art in two important ways: (1) they can take advantage of the standard chase, and (2) they yield new decidable tgd classes with data complexities that are complete for κ-ExpTime and κ-ExpSpace, for any $\kappa \geq 0$. This is obviously too high for transactional DBMS loads, but it allows us to address a much larger range of complex computational tasks over databases with the chase. Practical problems of this kind include ontology reasoning [14], database provenance computation [20], and querying databases with complex values [32]. We also note that checking our criteria is always dominated by Datalog reasoning, which is practically feasible and of lower complexity than some established criteria [16].

Our work brings up many follow-up questions. First, the new tgd classes are candidates for capturing their respective complexity classes (at least from PSpace upwards), but known proof techniques rely on non-saturating tgds [8]. Second, our techniques require a Datalog-first chase strategy, which is avoidable for sets and complex values [32]. It is open if similar approaches could apply in our setting. Third, our criteria can be broadened, e.g., the restriction to a single maximal-rank component in Section 6 can be relaxed.

Taking a wider view, a central methodological contribution of our work is the labelled dependency graph and its extensive use for analysing the internal structure of the standard chase. It can be seen as a surrogate for the more syntactic "lineage" of nulls that is available in the semi-oblivious chase – best exposed through the use of skolem terms [31] –, which has been extremely useful in studying that chase variant. It is exciting to ask how our method can be similarly useful in further studying the standard chase, e.g., to detect non-termination or to decide termination in new cases, and whether it can be refined in the style of termination checks based on materialisation or control flow analysis.

Acknowledgements. This work was partly supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in projects 389792660 (TRR 248, Center for Perspicuous Systems) and 390696704 (CeTI Cluster of Excellence); by the Bundesministerium für Bildung und Forschung (BMBF) in the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI); by BMBF and DAAD (German Academic Exchange Service) in project 57616814 (SECAI, School of Embedded and Composite AI); and by the Center for Advancing Electronics Dresden (cfaed).

REFERENCES

[1] Alfred V. Aho, Catriel Beeri, and Jeffrey D. Ullman. 1979. The Theory of Joins in Relational Databases. *ACM Trans. Database Syst.* 4, 3 (1979), 297–314.

[2] Alfred V. Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. 1979. Efficient Optimization of a Class of Relational Expressions. *ACM Trans. Database Syst.* 4, 4 (1979), 435–454. https://doi.org/10.1145/320107.320112

[3] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Swan Rocher, and Clément Sipieter. 2015. Graal: A Toolkit for Query Answering with Existential Rules. In *Proc. 9th Int. Web Rule Symposium (RuleML'15) (LNCS, Vol. 9202)*, Nick Bassiliades, Georg Gottlob, Fariba Sadri, Adrian Paschke, and Dumitru Roman (Eds.). Springer, 328–344.

[4] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. 2011. On rules with existential variables: Walking the decidability line. *Artificial Intelligence* 175, 9–10 (2011), 1620–1654.

[5] Catriel Beeri and Moshe Y. Vardi. 1984. A Proof Procedure for Data Dependencies. *J. ACM* 31, 4 (1984), 718–741.

[6] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the Chase. In *Proc. 36th Symposium on Principles of Database Systems (PODS'17)*, Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts (Eds.). ACM, 37–52.

[7] Michael Benedikt, Julien Leblay, and Efthymia Tsamoura. 2014. PDQ: Proof-driven Query Answering over Web-based Data. *Proc. VLDB Endowment* 7, 13 (2014), 1553–1556.

[8] Camille Bourgaux, David Carral, Markus Krötzsch, Sebastian Rudolph, and Michaël Thomazo. 2021. Capturing Homomorphism-Closed Decidable Queries with Existential Rules. In *Proc. 18th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'21)*, Meghyn Bienvenu, Gerhard Lakemeyer, and Esra Erdem (Eds.). 141–150.

[9] Marco Calautti, Georg Gottlob, and Andreas Pieris. 2022. Non-Uniformly Terminating Chase: Size and Complexity. In *Proc. 41st Symposium on Principles of Database Systems (PODS'22)*, Leonid Libkin and Pablo Barceló (Eds.). ACM, 369–378.

[10] Marco Calautti, Mostafa Milani, and Andreas Pieris. 2023. Semi-Oblivious Chase Termination for Linear Existential Rules: An Experimental Study. *Proc. VLDB Endow.* 16, 11 (2023), 2858–2870. https://doi.org/10.14778/3611479.3611493

[11] Andrea Calì, Georg Gottlob, and Andreas Pieris. 2012. Towards more expressive ontology languages: The query answering problem. *J. of Artif. Intell.* 193 (2012), 87–128.

[12] Andrea Calì, Domenico Lembo, and Riccardo Rosati. 2003. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. 22nd Symposium on Principles of Database Systems (PODS'03)*, Frank Neven, Catriel Beeri, and Tova Milo (Eds.). ACM, 260–271. https://doi.org/10.1145/773153.773179

[13] David Carral, Irina Dragoste, and Markus Krötzsch. 2017. Restricted Chase (Non)Termination for Existential Rules with Disjunctions. In *Proc. 26th Int. Joint Conf. on Artificial Intelligence (IJCAI'17)*, Carles Sierra (Ed.). ijcai.org, 922–928.

[14] David Carral, Irina Dragoste, Markus Krötzsch, and Christian Lewe. 2019. Chasing Sets: How to Use Existential Rules for Expressive Reasoning. In *Proc. 28th Int. Joint Conf. on Artificial Intelligence (IJCAI'19)*, Sarit Kraus (Ed.). ijcai.org, 1624–1631.

[15] David Carral, Lucas Larroque, Marie-Laure Mugnier, and Michaël Thomazo. 2022. Normalisations of Existential Rules: Not so Innocuous!. In *Proc. 19th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'22)*, Gabriele Kern-Isberner, Gerhard Lakemeyer, and Thomas Meyer (Eds.). 10 pages.

[16] Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. 2013. Acyclicity Notions for Existential Rules and Their Application to Query Answering in Ontologies. *J. of Artificial Intelligence Research* 47 (2013), 741–808.

[17] Anuj Dawar and Stephan Kreutzer. 2008. On Datalog vs. LFP. In *Proc. 35th Int. Colloquium on Automata, Languages, and Programming (ICALP'08); Part II (LNCS, Vol. 5126)*, Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz (Eds.). Springer, 160–171.

[18] Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. 2008. The Chase Revisited. In *Proc. 27th Symposium on Principles of Database Systems (PODS'08)*, Maurizio Lenzerini and Domenico Lembo (Eds.). ACM, 149–158.

[19] Alin Deutsch and Val Tannen. 2003. Reformulation of XML Queries and Constraints. In *Proc. 9th Int. Conf. on Database Theory (ICDT'03) (LNCS, Vol. 2572)*, Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani (Eds.). Springer, 225–241.

[20] Ali Elhalawati, Markus Krötzsch, and Stephan Mennicke. 2022. An Existential Rule Framework for Computing Why-Provenance On-Demand for Datalog. In *Proc. 2nd Int. Joint Conf. on Rules and Reasoning (RuleML+RR'22) (LNCS, Vol. 13752)*, Guido Governatori and Anni-Yasmin Turhan (Eds.). Springer, 146–163.

[21] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336, 1 (2005), 89–124.

[22] Sarah Alice Gaggl, Philipp Hanisch, and Markus Krötzsch. 2022. Simulating Sets in Answer Set Programming. In *Proc. 31st Int. Joint Conf. on Artificial Intelligence (IJCAI'22)*, Luc De Raedt (Ed.). ijcai.org, 2634–2640. https://doi.org/10.24963/ijcai.2022/365

[23] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2014. That's All Folks! LLUNATIC Goes Open Source. *PVLDB* 7, 13 (2014), 1565–1568.

[24] Tomasz Gogacz and Jerzy Marcinkowski. 2014. All-Instances Termination of Chase is Undecidable. In *Proc. 41st Int. Colloquium on Automata, Languages, and Programming (ICALP'14); Part II (LNCS, Vol. 8573)*, Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias (Eds.). Springer, 293–304.

[25] Tomasz Gogacz, Jerzy Marcinkowski, and Andreas Pieris. 2023. Uniform Restricted Chase Termination. *SIAM J. Comput.* 52, 3 (2023), 641–683. https://doi.org/10.1137/20M1377035
[26] Gösta Grahne and Adrian Onet. 2018. Anatomy of the Chase. *Fundam. Inform.* 157, 3 (2018), 221–270.
[27] Philipp Hanisch and Markus Krötzsch. 2024. Chase Termination Beyond Polynomial Time. arXiv:2403.16712 [cs.DB]
[28] Markus Krötzsch and Sebastian Rudolph. 2011. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In *Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11)*, Toby Walsh (Ed.). AAAI Press/IJCAI, 963–968.
[29] Markus Krötzsch, Maximilian Marx, and Sebastian Rudolph. 2019. The Power of the Terminating Chase. In *Proc. 22nd Int. Conf. on Database Theory (ICDT'19) (LIPIcs, Vol. 127)*, Pablo Barceló and Marco Calautti (Eds.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 3:1–3:17.
[30] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. 1979. Testing implications of data dependencies. *ACM Transactions on Database Systems* 4 (1979), 455–469. Issue 4.
[31] Bruno Marnette. 2009. Generalized Schema-Mappings: from Termination to Tractability. In *Proc. 28th Symposium on Principles of Database Systems (PODS'09)*, Jan Paredaens and Jianwen Su (Eds.). ACM, 13–22.
[32] Maximilian Marx and Markus Krötzsch. 2022. Tuple-Generating Dependencies Capture Complex Values. In *Proc. 25th Int. Conf. on Database Theory (ICDT'22) (LIPIcs, Vol. 220)*, Dan Olteanu and Nils Vortmeier (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:20. https://doi.org/10.4230/LIPIcs.ICDT.2022.13
[33] Michael Meier. 2014. The backchase revisited. *VLDB J.* 23, 3 (2014), 495–516. https://doi.org/10.1007/S00778-013-0333-Y
[34] Michaël Thomazo Michel Leclére, Marie-Laure Mugnier and Federico Ulliana. 2019. On Chase Termination for Linear Existential Rules. In *Proc. 22nd Int. Conf. on Database Theory (ICDT'19)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik.
[35] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. 2015. RDFox: A Highly-Scalable RDF Store. In *Proc. 14th Int. Semantic Web Conf. (ISWC'15), Part II (LNCS, Vol. 9367)*, Marcelo Arenas, Óscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul T. Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab (Eds.). Springer, 3–20.
[36] Sebastian Rudolph and Michaël Thomazo. 2015. Characterization of the Expressivity of Existential Rule Queries. In *Proc. 24th Int. Joint Conf. on Artificial Intelligence (IJCAI'15)*, Qiang Yang and Michael Wooldridge (Eds.). AAAI Press, 3193–3199.
[37] Sebastian Rudolph and Michaël Thomazo. 2016. Expressivity of Datalog Variants - Completing the Picture. In *Proc. 25th Int. Joint Conf. on Artificial Intelligence (IJCAI'15)*, Subbarao Kambhampati (Ed.). AAAI Press, 1230–1236.
[38] Jacopo Urbani, Markus Krötzsch, Ceriel J. H. Jacobs, Irina Dragoste, and David Carral. 2018. Efficient Model Construction for Horn Logic with VLog: System description. In *Proc. 9th Int. Joint Conf. on Automated Reasoning (IJCAR'18) (LNCS, Vol. 10900)*, Didier Galmiche, Stephan Schulz, and Roberto Sebastiani (Eds.). Springer, 680–688.
[39] Heng Zhang, Yan Zhang, and Jia-Huai You. 2015. Existential Rule Languages with Finite Chase: Complexity and Expressiveness. In *Proc. 29th AAAI Conf. on Artificial Intelligence (AAAI'15)*, Blai Bonet and Sven Koenig (Eds.). AAAI Press.

Received December 2023; revised February 2024; accepted March 2024