The Power of the Terminating Chase

Markus Krötzsch
TU Dresden, Germany
markus.kroetzsch@tu-dresden.de

Maximilian Marx
TU Dresden, Germany
maximilian.marx@tu-dresden.de

Sebastian Rudolph
TU Dresden, Germany
sebastian.rudolph@tu-dresden.de

Abstract

The chase has become a staple of modern database theory with applications in data integration, query optimisation, data exchange, ontology-based query answering, and many other areas. Most application scenarios and implementations require the chase to terminate and produce a finite universal model, and a large arsenal of sufficient termination criteria is available to guarantee this (generally undecidable) condition. In this invited tutorial, we therefore ask about the expressive power of logical theories for which the chase terminates. Specifically, which database properties can be recognised by such theories, i.e., which Boolean queries can they realise? For the skolem (semi-oblivious) chase, and almost any known termination criterion, this expressivity is just that of plain Datalog. Surprisingly, this limitation of most prior research does not apply to the chase in general. Indeed, we show that standard–chase terminating theories can realise queries with data complexities ranging from PTime to non-elementary that are out of reach for the terminating skolem chase. A "Datalog-first" standard chase that prioritises applications of rules without existential quantifiers makes modelling simpler – and we conjecture: computationally more efficient. This is one of the many open questions raised by our insights, and we conclude with an outlook on the research opportunities in this area.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Theory of computation → Complexity theory and logic; Theory of computation → Database constraints theory; Theory of computation → Logic and databases

Keywords and phrases Existential rules, Tuple-generating dependencies, all-instances chase termination, expressive power, data complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.3

Category Invited Talk

Funding This work is partly supported by the German Research Foundation (DFG) in CRC 248 (Perspicuous Systems), CRC 912 (HAEC), and Emmy Noether grant KR 4381/1-1; and by the European Research Council (ERC) Consolidator Grant 771779 (DeciGUT).

Acknowledgements We thank David Carral for his comments on an earlier version of this paper.

1 Introduction

Forty years ago, the first versions of the *chase* algorithm were developed, initially as a way of deciding implication problems of specific classes of dependencies [26, 2], and soon thereafter as a proof system for the more general (and undecidable) case of *tuple-generating* and *equality-generating dependencies* (TGDs and EGDs) [6, 7]. As of today, the chase is understood as an iterative method for constructing *universal models* of Horn logic theories [19], which corresponds to the computation of most general solutions to logical entailment problems. Such

22nd International Conference on Database Theory (ICDT 2019).
Editors: Pablo Barcelo and Marco Calautti; Article No. 3; pp. 3:1–3:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

universal solutions have many natural applications in data integration, query optimisation, data exchange, and query answering under constraints. The re-discovery of TGDs as *existential rules*, and their application to ontological modelling and knowledge representation has motivated many further studies [3, 11, 15].

Of course, our practical interest is usually in cases where the chase terminates and the resulting model is therefore finite, corresponding to a database that has been "repaired" to satisfy the given dependencies. Unfortunately, chase termination is undecidable, no matter if we ask for the termination on a particular database [7] or for "universal" termination on all databases [22, 23]. Nonetheless, practical implementations of chase procedures have been successfully applied in many contexts [21, 4, 9, 8, 35]. It is then up to the user to provide terminating inputs, and this is facilitated by many decidable sufficient criteria for ensuring chase termination, which have been developed for this purpose [20, 27, 3, 25, 15, 12].

This brings up a natural question: given an efficient practical implementation of some variant of the chase, what computational problems can we solve with it? That is: what is the expressive power of logical theories for which the chase terminates? To answer this, we first need to clarify what the *input* is: the theory and database, or merely the database while the theory is fixed? These two views can be associated with traditional perspectives from formerly separated and now converging fields of computer science [31]. The first perspective is that of knowledge representation, where logical theories ("ontologies") are exchanged and combined, and reasoning over such theories is necessary to access the encoded knowledge.

The second perspective is common in database theory, where theories are comparatively small and static, whereas databases are large and dynamic. This is also reflected in many current chase implementations, which treat logical theories as "programs," usually in a tool-specific format that prevents any kind of interchange, while supporting data-level interoperability with standard sources (such as relational DBMS or RDF stores). The data-centric view is also taken by most studies on termination criteria, which aim to identify theories for which the chase will terminate with *every* database instance.

We can therefore specify our question as follows: what is the expressivity of logical theories for which the chase terminates on all database instances? Surprisingly little is known about this, and what we can conclude from previous results is thoroughly disappointing: if our chase variant is the skolem (a.k.a. semi-oblivious) chase, expressivity is essentially limited to traditional Datalog [27, 25, 36]. The vast majority of known termination criteria applies to the skolem chase [15], and is therefore facing the same limitations. In spite of the significant challenges that existential quantifiers introduce to rule-based reasoning, it seems that our current approaches to ensuring decidability cannot offer substantial advantages regarding the problems that one can solve in practice.

Surprisingly, this apparent weakness of the chase is specific to the skolem chase and the termination criteria that are tailored to this variant. We show this by demonstrating

1. that there are PTime decision problems that are not expressible in Datalog, and hence neither via terminating skolem chase, but that can be solved in the terminating standard (a.k.a. restricted) chase, and
2. that the terminating standard chase can solve decision problems of non-elementary (data) complexity, whereas Datalog and the skolem chase have PTime data complexity.

Therefore, the standard chase is superior to the skolem chase regarding tractable as well as highly complex computations. Each implementation of this algorithm inherits this power – as long as one is willing to give up the restriction to skolem-chase terminating theories.

As another contribution, we investigate a particularly natural class of chase strategies, where Datalog rules (containing no existential quantification) are always applied before any true existential rule. This *Datalog-first* strategy simplifies the construction of universally terminating theories, and in fact is the only variant of the chase for which we could solve decision problems as in (1) in polynomial time. Although we give an encoding of (1) that terminates in the standard chase, this might require exponentially many steps (in spite of the query describing a property of databases recognizable in PTime).

A major consequence of our study is that focussing research activities related to chase termination on the skolem chase is arguably too restrictive, since one relinquishes significant expressive power when doing so. This insight is complemented by recent empirical studies proving the standard chase almost always more efficient in practical implementations [8, 35]. This might be surprising, given that a standard chase step is computationally more demanding [23]; but the reduction in redundant computation seems to compensate for this cost even in cases where termination does not rely on it. Shifting the focus away from the skolem chase also reveals how little we know about more complex chase variants, and raises a number of open questions for future research.

After a quick overview of preliminary definitions (Section 2), we review several important chase variants (Section 3), and the state of the art regarding their termination (Section 4). Thereafter, we make the previously established expressive limits of the skolem chase explicit (Section 5). We then continue to demonstrate that these limits can be overcome both on queries in PTime (Section 6) and on queries of considerably higher complexity (Section 7). We conclude with an overview of open questions and conjectures (Section 8).

2 Preliminaries

We briefly introduce the necessary concepts and terminology. For a more thorough introduction, we refer to [1, 14]. We construct expressions from countably infinite, mutually disjoint sets $\mathbf{V}$ of *variables*, $\mathbf{F}$ of *(skolem) function symbols*, $\mathbf{N}$ of *labelled nulls*, and $\mathbf{R}$ of *relation names*. Each function or relation name $s \in \mathbf{F} \cup \mathbf{R}$ has an *arity* $\mathsf{ar}(s) \geq 0$. Function symbols of arity 0 are *constants*, and we set $\mathbf{C} \coloneqq \{c \in \mathbf{F} \mid \mathsf{ar}(c) = 0\}$. *Terms* are either elements of $\mathbf{V} \cup \mathbf{N} \cup \mathbf{C}$, or, recursively, expressions $f(t_1, \ldots, t_n)$ with $f \in \mathbf{F}$, $\mathsf{ar}(f) = n$, and $t_1, \ldots, t_n$ terms. We generally use $\boldsymbol{t}$ to denote a list $t_1, \ldots, t_{|\boldsymbol{t}|}$ of terms, and similar for special types of terms. An *atom* is an expression $r(\boldsymbol{t})$ with $r \in \mathbf{R}$, $\boldsymbol{t}$ a list of terms, and $\mathsf{ar}(r) = |\boldsymbol{t}|$. *Ground* terms or atoms contain neither variables nor nulls.

Rules and queries. An *existential rule* (or just *rule*) ρ is a formula

$$\rho = \forall \boldsymbol{x}, \boldsymbol{y}.\ \varphi[\boldsymbol{x}, \boldsymbol{y}] \to \exists \boldsymbol{z}.\ \psi[\boldsymbol{y}, \boldsymbol{z}], \tag{1}$$

where φ and ψ are conjunctions of atoms using only terms from $\mathbf{C}$ or from the mutually disjoint lists of variables $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \subseteq \mathbf{V}$. We call φ the *body* (denoted $\mathsf{body}(\rho)$), ψ the *head* (denoted $\mathsf{head}(\rho)$), and $\boldsymbol{y}$ the *frontier* of ρ. We often treat conjunctions of atoms like sets, and omit the universal quantifiers in rules. A rule is *Datalog* if it has no existential quantifiers. A *conjunctive query* (CQ) $q[\boldsymbol{x}]$ with free variables $\boldsymbol{x}$ is a formula $\exists \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}]$, where φ is a conjunction of atoms using only constants and variables from $\boldsymbol{x} \cup \boldsymbol{y}$. A *boolean conjunctive query* (BCQ) is a CQ without free variables. Since CQ answering and BCQ answering are polynomially reducible to one another [3], we restrict our attention to BCQs.

Databases and morphism. A *database* $\mathcal{I}$ is a set of atoms without variables. A *concrete database* $\mathcal{D}$ is a finite database without function symbols, except for constants. Given a set of atoms $\mathcal{A}$ and database $\mathcal{I}$, a *homomorphism* $h : \mathcal{A} \to \mathcal{I}$ is a function that maps the terms occurring in $\mathcal{A}$ to (the variable-free) terms occurring in $\mathcal{I}$, such that: (i) for all $f \in \mathbf{F}$: $h(f(\boldsymbol{t})) = f(h(\boldsymbol{t}))$; (ii) for all $r \in \mathbf{R}$: if $r(\boldsymbol{t}) \in \mathcal{A}$, then $r(h(\boldsymbol{t})) \in \mathcal{I}$, where $h(\boldsymbol{t})$ is the list of h-images of the terms $\boldsymbol{t}$. A homomorphism h is *strong* if $r(\boldsymbol{t}) \in \mathcal{A} \iff r(h(\boldsymbol{t})) \in \mathcal{I}$ for all $r \in \mathbf{R}$, and an *embedding* if it is strong and injective.

Semantics of rules and queries. A match of a rule ρ in a database $\mathcal{I}$ is a function h from the universally quantified variables of ρ to $\mathcal{I}$ that restricts to a homomorphism $\mathsf{body}(\rho) \to \mathcal{I}$.[1] A match h of ρ in $\mathcal{I}$ is *satisfied* if there is a homomorphism $h' : \mathsf{head}(\rho) \to \mathcal{I}$ that agrees with h on all frontier variables. Rule ρ is *satisfied* by $\mathcal{I}$, written $\mathcal{I} \models \rho$, if every match of ρ in $\mathcal{I}$ is satisfied. A set of rules Σ is satisfied by $\mathcal{I}$, written $\mathcal{I} \models \Sigma$, if $\mathcal{I} \models \rho$ for all $\rho \in \Sigma$. We may treat a concrete database $\mathcal{D}$ as sets of rules with empty bodies (also called *facts*), and write, e.g., $\mathcal{I} \models \mathcal{D}, \Sigma$ to express that $\mathcal{I} \models \Sigma$ and $\mathcal{D} \subseteq \mathcal{I}$. In this case, $\mathcal{I}$ is a *model* of Σ and $\mathcal{D}$. A BCQ $q = \exists \boldsymbol{x}.\varphi[\boldsymbol{x}]$ is *satisfied* by a database $\mathcal{I}$, written $\mathcal{I} \models q$, if there is a homomorphism $\varphi[\boldsymbol{x}] \to \mathcal{I}$; it is *entailed* by a concrete database $\mathcal{D}$ and rule set Σ, written $\mathcal{D}, \Sigma \models q$, if $\mathcal{I} \models q$ for every $\mathcal{I}$ with $\mathcal{I} \models \mathcal{D}, \Sigma$ (note that for empty rule sets, satisfaction coincides with entailment). Since homomorphisms are closed under composition, the existence of homomorphisms $q \to \mathcal{I}$ and $\mathcal{I} \to \mathcal{J}$ implies that q is satisfied by $\mathcal{J}$: the set of databases that satisfy a BCQ is *closed under homomorphisms*.

Expressivity. When discussing expressivity of query languages, it is convenient to have an abstract notion of (boolean) queries. An *abstract query* over a given finite signature $\mathbf{R}^{\mathrm{EDB}} \subseteq \mathbf{R}$ of so-called *extensional database relations* is a set $\mathfrak{D}$ of concrete databases over $\mathbf{R}^{\mathrm{EDB}}$. We say that a given set of rules Σ and BCQ q *realise* $\mathfrak{D}$ if for every database $\mathcal{D}$ over $\mathbf{R}^{\mathrm{EDB}}$ holds $\mathcal{D}, \Sigma \models q$ exactly if $\mathcal{D} \in \mathfrak{D}$. Note that Σ and q can make use of other (so-called *intensional database*) relations not in $\mathbf{R}^{\mathrm{EDB}}$. This notion of query realisation allows us to focus on input databases which are "well-formed" in terms of the relation names used.

For an arbitrary set of rules Σ, checking $\mathcal{D}, \Sigma \models q$ is undecidable, however, the set $\{\mathcal{D} \mid \mathcal{D}, \Sigma \models q\}$ is recursively enumerable and closed under homomorphisms. In fact, for every homomorphism-closed, recursively enumerable abstract query $\mathfrak{D}$, there exist Σ and q that realise $\mathfrak{D}$ [32].

3 Universal Models and the Chase

BCQ entailment (and CQ answering) can be solved by computing *universal models* [19]. A model $\mathcal{I}$ of a set of rules Σ is universal if it admits a homomorphism $h : \mathcal{I} \to \mathcal{J}$ to every model $\mathcal{J}$ of Σ. Due to closure under homomorphisms, BCQs that are entailed by a universal model are entailed by every model. Since the converse is also true, universal models characterise BCQ entailment.[2] For this reason (among others), many algorithms for computing universal models have been developed. Their basic approach, known as the *chase*, is to construct such models bottom-up by applying rules to facts. We consider several variants of this approach: the *standard* (a.k.a. *restricted*) *chase* [20], the *skolem* (a.k.a. *semi-oblivious*) *chase* [27], and the *core chase* [19].

[1] If all frontier variables occur in body atoms (i.e., the rule is *safe*), h is a homomorphism $\mathsf{body}(\rho) \to \mathcal{I}$ itself; the chosen definition accommodates the possibility of unsafe rules.

[2] We remark that universal models are generally neither unique, nor the only models that characterise BCQ entailment (see [13] for some discussion).

We first define the standard and skolem chases. For a rule ρ of form (1), the *skolemised rule* $\mathrm{sk}(\rho)$ is $\varphi[\boldsymbol{x},\boldsymbol{y}] \to \psi[\boldsymbol{y},\boldsymbol{t_z}]$, where $\psi[\boldsymbol{y},\boldsymbol{t_z}]$ is obtained from $\psi[\boldsymbol{y},\boldsymbol{z}]$ by replacing each variable $z \in \boldsymbol{z}$ with a skolem term $f(\boldsymbol{y})$ using a fresh skolem function symbol f. Notions that were defined for existential rules naturally extend to skolemised rules. For uniformity, the next definition also treats skolemised rules as formulas of the form (1), where $\boldsymbol{z}$ is empty and ψ may contain functions instead.

▶ **Definition 1.** *A* chase sequence *for a concrete database $\mathcal{D}$ and a set of existential or skolemised rules Σ is a potentially infinite sequence $\mathcal{D}^0, \mathcal{D}^1, \ldots$ such that*

(1) *$\mathcal{D}^0 = \mathcal{D}$;*

(2) *for every $\mathcal{D}^i$ with $i \geq 0$, there is a match h for some rule $\rho = \varphi[\boldsymbol{x},\boldsymbol{y}] \to \exists \boldsymbol{z}.\psi[\boldsymbol{y},\boldsymbol{z}] \in \Sigma$ in $\mathcal{D}^i$ such that*

  a. *h is an unsatisfied match in $\mathcal{D}^i$ (i.e., h cannot be extended to a homomorphism $\psi \to \mathcal{D}^i$), and*

  b. *$\mathcal{D}^{i+1} = \mathcal{D}^i \cup \psi[h'(\boldsymbol{y}), h'(\boldsymbol{z})]$, where $h' : \psi \to \mathcal{D}^{i+1}$ is such that $h'(y) = h(y)$ for all $y \in \boldsymbol{y}$, and for all $z \in \boldsymbol{z}$, $h'(z) \in \mathbf{N}$ is a distinct labelled null that does not occur in $\mathcal{D}^i$ (in particular, h is a satisfied match in $\mathcal{D}^{i+1}$).*

(3) *if h is a match for a rule $\rho \in \Sigma$ and $\mathcal{D}^i$ $(i \geq 0)$, then there is $j > i$ such that h is satisfied in $\mathcal{D}^j$ (*fairness*).*

The chase *for such a chase sequence is the database $\bigcup_{i\geq 0} \mathcal{D}^i$.*

Note that this definition allows individual rule applications to occur in any (fair) order, and in particular does not require that all matches for a rule are processed together. This generality makes sense to cover all current implementations, some of which deploy parallelised, streaming rule applications that might result in such an interleaved derivation order [30].

Given a set Σ of existential rules, a chase for Σ is called *standard chase*, and a chase for $\mathrm{sk}(\Sigma)$ is called *skolem chase*. A *Datalog-first chase* is a standard chase where non-Datalog rules are applied in step (2) only if all matches for Datalog rules in $\mathcal{D}^{i-1}$ are satisfied. Standard and Datalog-first chases might not be unique, due to the dependence of condition (2.a) on the order of rule applications, i.e., on the chase *strategy*.

▶ **Example 2.** Consider the concrete database $r(a,b)$ and the rules

$$r(x,y) \to \exists v.r(y,v) \tag{2}$$

$$r(x,y) \to r(y,y) \tag{3}$$

In the standard chase, we can apply rule (2) with the match $\{x \mapsto a, y \mapsto b\}$ to derive $r(b,n_1)$, where n_1 is a new null. Applying (3) to $\{x \mapsto a, y \mapsto b\}$ yields $r(b,b)$, which would have blocked the first rule application if it had been computed earlier. Depending on the order of further rule applications in the given strategy, the standard chase might terminate after arbitrarily many steps, or fail to terminate altogether. In contrast, the Datalog-first chase prioritises (3), and therefore terminates with the model $\{r(a,b), r(b,b)\}$. Regarding the entailed BCQs, all of these results (including the infinite r-chain with loops) are equivalent.

The Datalog-first chase restricts to a sensible class of strategies, but it does not free the chase from its dependence on a selected strategy. For example, standard and Datalog-first chases coincide whenever there are no Datalog rules, which is easy to achieve by extending rule heads with redundant existential statements, e.g., we could replace (3) by $r(x,y) \to \exists v.r(y,y) \wedge r(x,v)$. In contrast, the skolem chase is always unique, since (2.a) is equivalent to $\psi[h(\boldsymbol{y})] \not\subseteq \mathcal{D}^{i-1}$ in this case.

▶ **Example 3.** Skolemising Example 2, rule (2) turns into $r(x, y) \to r(y, f(y))$ for some skolem function f. The skolem chase is the infinite database $\{r(a, b), r(b, b), r(b, f(b)), r(f(b), f(b)), r(f(b), f(f(b))), \ldots\}$.

Further variations of these chases have been defined in the literature [8]. For example, the *parallel chase* computes $\mathcal{D}^{i+1}$ from $\mathcal{D}^i$ by considering all matches (w.r.t. $\mathcal{D}^i$) in step (2). $\mathcal{D}^{i+1}$ then contains new derivations from many rule applications, even if some of them could have prevented the application of others. The *1-parallel chase* is similar, but only considers the matches of one rule in each step. These modifications lead to variants of the standard and Datalog-first chase, but do not affect the skolem chase.

An important extension of the parallel standard chase is the *core chase*, where the database is reduced to a core after each derivation step [19]. A database $\mathcal{I}$ is a *core* if every homomorphism $h : \mathcal{I} \to \mathcal{I}$ is an embedding. Given a database $\mathcal{J}$, a *core of* $\mathcal{J}$ is a core obtained by restricting $\mathcal{J}$ to its image under some appropriate homomorphism $h : \mathcal{J} \to \mathcal{J}$ [24, 5]. Every finite database $\mathcal{D}$ has a unique core (up to isomorphism),[3] which we denote by $\mathsf{core}(\mathcal{D})$.

▶ **Definition 4.** *A* core chase sequence *for a concrete database $\mathcal{D}$ and a set of existential rules Σ is a maximal sequence $\mathcal{D} = \mathcal{D}^0, \mathcal{D}^1, \ldots,$ such that D^i is not isomorphic to D^{i+1} and $\mathcal{D}^{i+1} = \mathsf{core}(\Sigma(\mathcal{D}^i))$, where $\Sigma(\mathcal{D}^i)$ is the result of applying all rules $\rho \in \Sigma$ that have a match for $\mathcal{D}^i$ as in step (2) of Definition 1. If this sequence is finite, then its final element D^ℓ is the* core chase*, which is then unique up to isomorphism.*

The key advantage of this more complex algorithm is that it characterises when a set of rules admits a finite universal model over a given database:

▶ **Theorem 5** ([19])**.** *A concrete database $\mathcal{D}$ and a set of existential rules Σ admit a finite universal model if and only if the core chase terminates (producing such a model).*

4 Chase Termination

In practice, we are particularly interested in cases where the chase produces a finite universal model. In this section, we discuss this situation and review several criteria for recognising it effectively. We start by defining the most important types of termination.

Let Σ be a set of existential rules and a concrete database $\mathcal{D}$. Σ has an *all-strategies terminating* standard chase on $\mathcal{D}$ if all chase sequences for Σ and $\mathcal{D}$ are finite. Adopting notation of Grahne and Onet [23], we write $\mathsf{CT}^{\mathsf{std}}_{\mathcal{D}\forall}$ for the class of all such rule sets. Analogously, $\mathsf{CT}^{\mathsf{dlf}}_{\mathcal{D}\forall}$ denotes the class of all *Datalog-first terminating* rule sets. These notions can be generalised to require *all-instances* termination:[4] $\mathsf{CT}^{\mathsf{std}}_{\forall\forall} = \bigcap_{\mathcal{D}} \mathsf{CT}^{\mathsf{std}}_{\mathcal{D}\forall}$ and $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall} = \bigcap_{\mathcal{D}} \mathsf{CT}^{\mathsf{dlf}}_{\mathcal{D}\forall}$. For the skolem chase and for the core chase, the chosen strategy does not affect termination, hence we simplify notation and write $\mathsf{CT}^{\mathsf{sk}}_{\mathcal{D}}$ and $\mathsf{CT}^{\mathsf{sk}}_{\forall}$, respectively, and similarly for the core chase. It is known that $\mathsf{CT}^{\mathsf{sk}}_{\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{core}}_{\forall}$ [23], and it is similarly easy to see that we also have $\mathsf{CT}^{\mathsf{std}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{core}}_{\forall}$.

Termination is generally undecidable, often not even recursively enumerable [22, 23],[5] but many sufficient criteria have been proposed. Most existing works give criteria for inclusion in

[3] Infinite databases can have several non-isomorphic cores, or none at all [13].

[4] Note that this includes instances $\mathcal{D}$ that do not only contain extensional database relations, i.e., termination is also required if the input database is not "well-formed."

[5] For the Datalog-first chase, one can reduce from the undecidable termination problems for the standard chase by extending Datalog rules with irrelevant existential quantifiers.

$\mathsf{CT}_{\forall}^{\mathsf{sk}}$ [20, 27, 28, 25, 3, 15], and some can also be used for data-dependent classes $\mathsf{CT}_{\mathcal{D}}^{\mathsf{sk}}$ [28, 15]. Comparatively few works propose criteria that exploit the better termination behaviour of the standard chase [19, 28, 12]. We give a more detailed overview below.

Recent works established the decidability of termination for syntactically restricted classes of rules: $\mathsf{CT}_{\forall}^{\mathsf{sk}}$ is decidable on *sticky rules* [10], while $\mathsf{CT}_{\forall\forall}^{\mathsf{std}}$ is decidable on linear rules [29] and also on guarded rules [34]. An even stronger notion than all-instances termination is *k-boundedness*, which requires that any fact in the chase can be derived by at most k rule applications. This criterion is known to be decidable for skolem and standard chase [18].

4.1 Termination of the Skolem Chase

In general, the question "$\Sigma \in \mathsf{CT}_{\mathcal{D}}^{\mathsf{sk}}$?" is clearly semi-decidable by simply running the chase, i.e., $\mathsf{CT}_{\mathcal{D}}^{\mathsf{sk}}$ is recursively enumerable. Marnette observed that all-instance termination can be reduced to this special case by considering what he called the *critical instance* [27].

▶ **Definition 6.** *Let Σ be a rule set, and denote by* $\mathrm{const}(\Sigma)$ *the sets of constants occurring in Σ. The* critical instance *for Σ is the concrete database $\mathcal{D}^*$ consisting of all atoms of form $r(\boldsymbol{t})$, where r is a relation name occurring in Σ and $\boldsymbol{t}$ is a list of constants from $\{*\} \cup \mathrm{const}(\Sigma)$, with $*$ a fresh constant. By a slight abuse of notation, we write $\mathsf{CT}_{\mathcal{D}^*}^{\mathsf{sk}}$ for the class of all rule sets for which the skolem chase on their respective critical instance terminates.*

▶ **Theorem 7** ([27]). $\mathsf{CT}_{\mathcal{D}^*}^{\mathsf{sk}} = \mathsf{CT}_{\forall}^{\mathsf{sk}}$.

Therefore, $\mathsf{CT}_{\forall}^{\mathsf{sk}}$ is recursively enumerable, and, moreover, any technique for establishing skolem-chase termination on a specific instance can be used to establish all-instance termination. Many such techniques have been proposed. We can readily turn the chase itself into a decidable termination criterion if we ensure that the computation will halt even if the chase does not terminate. For example, we can stop the skolem chase when a *cyclic skolem term* occurs, i.e., a term of the form $f(t_1, \ldots, t_n)$ where f also occurs in some t_i. This termination criterion was called *MFA* (*model-faithful acyclicity*) and yields one of the largest decidable classes of skolem-chase terminating rules known today [15].

MFA is decidable but 2ExpTime-complete, and the chase might become double exponential before a cyclic term occurs. Many easier-to-check criteria are obtained by abstracting from the exact chase sequence and adapting the notion of *cycle* accordingly. If we identify terms that use the same outermost function symbol (equivalently: replace skolem terms by constants), we obtain *MSA* (*model-summarising acyclicity*), which is ExpTime-complete [15]. *Super-weak acyclicity* (SWA) [27] and *joint acyclicity* (JA) [25] replace terms by *places* and *positions*, respectively, both of which describe terms based on the general shape of inferred facts they might occur in. These possible facts can be (over-)estimated in polynomial time, and both criteria are PTime-complete. Even simpler – namely NL-complete – is *weak-acyclicity* (WA), which over-estimates term propagation by solving a reachability problem [20]. In terms of generality, the notions form a total order:

$$\mathrm{WA} \subset \mathrm{JA} \subset \mathrm{SWA} \subset \mathrm{MSA} \subset \mathrm{MFA} \subset \mathsf{CT}_{\forall}^{\mathsf{sk}}$$

Another way of abstracting MFA is to consider dependencies among rules instead of propagation of terms, i.e., we ask whether the conclusion produced by one rule might make another rule applicable. Termination is guaranteed if we thus obtain an *acyclic graph of rule dependencies* [3]. While subsumed by MFA, this NP-complete criterion is incomparable to other term-based notions [15]. Various kinds of dependencies of rules can be used to decompose rule sets into *strata*, which can then be analysed independently for termination [19, 28, 3].

Other sufficient criteria for skolem-chase termination were proposed. However, to the best of our knowledge, our listing includes all of the best-performing basic proposals in terms of complexity/expressivity trade-off.[6]

4.2 Termination of the Standard and Core Chase

Much less is known for the standard chase. Skolem chase termination yields a sufficient condition, since $\mathsf{CT}^{\mathsf{sk}}_{\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, but very few criteria take advantage of the standard chase's stronger inclination to terminate. Deutsch et al. define rule dependencies that are specific to the standard chase [19], and obtain a criterion for all-instances termination of the standard chase under *some* strategies. A corrected version that works for all strategies was proposed by Meier et al. [28].

These early dependency-based termination conditions were combined with weak acyclicity, i.e., term-level cycle detection remained confined to conditions for $\mathsf{CT}^{\mathsf{sk}}_{\forall}$. The difficulty is that the critical instance can no longer be used to detect universal termination, since the standard chase always terminates on this instance. More generally, the set of databases on which the standard chase with a given rule set terminates is no longer closed under homomorphisms – adding more facts may lead to termination.

Carral et al. recently proposed a way of approaching this problem [12]. Their notions of *restricted JA* (RJA) and *restricted MFA* (RMFA) show membership in $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$, i.e., for arbitrary instances and Datalog-first strategies, while also covering rule sets that are not in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ (or $\mathsf{CT}^{\mathsf{sk}}_{\forall}$). The essential idea for RMFA is to perform a modified skolem chase that, like MFA, starts from the critical instance and checks for cyclic terms. However, a match of a skolem rule is only applied after verifying that the match is not necessarily satisfied in all Datalog-first chase sequences. To this end, rule applications are retraced to find the most general set of facts from which the considered rule match can follow – a "universal premise" of the match that has a homomorphism into any chase where this match occurs. The Datalog-first strategy is honoured by further closing these facts under the given Datalog rules. If the match is satisfied by this Datalog-closed, universal premise, then the match is *blocked* and will not be used. The approach for RJA uses a similar block check to restrict the estimated propagation of terms along positions in JA.

It is easy to generalise RMFA and RJA to arbitrary (not necessarily Datalog-first) strategies by simply omitting the closure under Datalog rules. However, the wider practical applicability shown empirically by Carral et al. often hinges on the presence of Datalog rules that make the use of other existential rules obsolete.

For the core chase, due to Theorem 5, termination criteria correspond to general criteria for showing that a rule set has a finite universal model. We are not aware of any specific criteria that were proposed for this case while covering cases that are not in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.

5 The Weakness of the Terminating Skolem Chase

Already when Marnette first proposed the skolem chase, he observed an important feature that is tied to its all-instances termination: the size of the chase of any rule set in $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is polynomial in the size of the underlying concrete database [27]. It follows that the data complexity of BCQ answering over $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is in PTime, but also that $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is inherently limited

[6] There can be no "most general termination criterion" for any complexity class that is closed under finite variations, so "best-performing" can only refer to empirical coverage of rule sets found in practice [15].

in terms of its expressive power. In this section, we elaborate on this limitation and relate it to more recent insights on the expressivity limits of Datalog. Let us start by explaining Marnette's result:

▶ **Theorem 8** ([27]). *For every $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ and concrete database $\mathcal{D}$, the skolem chase over Σ and $\mathcal{D}$ is polynomial in the size of $\mathcal{D}$. The data complexity of BCQ entailment over $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is* PTime*-complete.*

Proof. We expand on the idea underlying the use of the critical instance in Theorem 7. Let $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D})$ denote the skolem chase over Σ and $\mathcal{D}$. For any concrete database $\mathcal{D}$, let $h : \mathcal{D} \to \mathcal{D}^*$ be the unique mapping that (i) is identical on const(Σ), (ii) maps all other constants to $*$, and (iii) satisfies $h(f(\boldsymbol{t})) = f(h(\boldsymbol{t}))$ for all $f \in \mathbf{F}$ with $\mathsf{ar}(f) \geq 1$. Then h extends to a mapping $h' : \mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D}) \to \mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D}^*)$ satisfying (i)–(iii), as is easy to show by induction along the length of the chase sequence. In particular, $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D})$ contains only terms for which a term of the same nesting structure occurs in $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D}^*)$. The number k of distinct terms in $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D}^*)$ is finite, and there is a largest number ℓ of occurrences of constants in these terms. Therefore, if $\mathcal{D}$ contains n distinct nulls and constants, and relation names of arity $\leq a$, then $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D})$ contains $\leq kn^\ell$ terms and $\leq (kn^\ell)^a$ facts, which is polynomial in n. The number of possible matches of any rule body likewise is polynomial in n, and $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D})$ can therefore be computed in polynomial time. PTime-hardness follows from the known data complexity of Datalog. ◀

Theorem 8 already establishes that rule sets in $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ cannot express properties that are not in PTime, such as defining an ExpTime-hard set of concrete databases. However, we can obtain even tighter limits. Krötzsch and Rudolph define a transformation from jointly acyclic rule sets (a subset of $\mathsf{CT}^{\mathsf{sk}}_{\forall}$) to Datalog that preserves entailment of (similarly rewritten) BCQs [25]. Zhang et al. observed that this idea generalises to all of $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ and that BCQ rewriting is not necessary if we allow for a set of weakly acyclic "output" rules on top of Datalog [36]. In summary, we find that $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ has the same expressivity as Datalog:

▶ **Theorem 9.** *For every $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ and BCQ q, there is a set of Datalog rules Σ' and BCQ q' such that $\{\mathcal{D} \mid \mathcal{D}, \Sigma \models q\} = \{\mathcal{D} \mid \mathcal{D}, \Sigma' \models q'\}$.*

Proof sketch. As observed in the proof of Theorem 8, the size of terms occurring in any chase of Σ is bounded. We can therefore encode terms as bounded-length lists of elements (that occur as leaves in the term tree), new auxiliary constants (to encode function symbols of arity > 1), and the special constant $\square$ (to fill unused positions in a list). If the maximal arity of skolem functions is k and the maximal nesting depth of functions in the critical-instance chase is ℓ, then each term can be represented as a list of $k^\ell + k^{\ell-1} + 1$ elements (corresponding to the total number of leaves and inner nodes of a tree of depth ℓ and branching factor k). For example, if $k = 2$ and $\ell = 2$, then the fact $p(f(a, g(b)))$ can be "flattened" to $\hat{p}(f, a, \square, \square, g, b, \square)$. It is not hard to modify all (skolemised) rules accordingly. ◀

It is clear that $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ does not capture PTime (since existential rules can only express properties that are preserved under homomorphisms), but Theorem 9 further confines $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ to the expressiveness boundaries of Datalog, which cannot even express every homomorphism-closed Boolean query in PTime. Dawar and Kreutzer show that the following query cannot be realised by Datalog [17]:

▶ **Definition 10.** *Consider the following decision problem:*

Input: A directed graph G with two distinguished vertices s and t
Question: Is G cyclic, or is there a simple path from s to t of length $2^{2^{n^2}}$ for some $n \in \mathbb{N}$?

The DK query $\mathfrak{D}_{\mathrm{DK}}$ *is the abstract Boolean query containing exactly those concrete databases that encode an instance of this decision problem using a binary relation* `edge` *and constant symbols* `s` *and* `t`*.*

$\mathfrak{D}_{\mathrm{DK}}$ is closed under homomorphisms: homomorphisms preserve cycles, and simple paths are either preserved or mapped to sub-structures with cycles. Moreover, the DK query can be checked in polynomial time, where we note that the length of any simple path is at most linear in the size of the input. Therefore, since the DK query is not expressible in Datalog [17], Theorem 9 implies that there are homomorphism-closed PTime queries that cannot be expressed in $\mathsf{CT}^{\mathsf{sk}}_{\forall}$:

▶ **Theorem 11.** *The DK query* $\mathfrak{D}_{\mathrm{DK}}$ *is homomorphism-closed and in* PTime*, yet it is not realised by any* $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ *and BCQ* q*.*

6 Beyond the Skolem Chase

Surprisingly, the mechanisms that limit the expressive power of $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ are specific to the skolem chase. Other chase variants are not confined in such ways, and can break both the PTime-barrier of Theorem 8 and surpass the expressive power of Datalog within PTime. In this section, we focus on the latter aspect by discussing several techniques of expressing the Dawar-Kreutzer query of Definition 10. We start with a result for the Datalog-first chase:

▶ **Theorem 12.** *There is a rule set* $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ *and BCQ* q *that realise the DK query. Moreover, the Datalog-first chase on* Σ *is polynomial in the size of the input database.*

Note that Theorem 12 establishes two independent results: (1) that the terminating Datalog-first chase is more expressive than the terminating skolem chase even on polynomial-time queries; and (2) that it will terminate in polynomial time in spite of this increased expressivity. For the standard chase, we will also establish property (1), but we conjecture that (2) cannot be attained.

Proof of Theorem 12. Inspired by a technique from Rudolph and Thomazo [33], we provide a rule set $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ with the required properties in Figure 1, where the corresponding BCQ is `goal` (a query with a nullary relation name). The first group of rules (4)–(6) computes the transitive closure `path` of `edge`, and derives `goal` if there is a cycle.

Rules (7)–(16) check for the other condition of the DK query by measuring the length of paths from `s` to `t`. We define an initial `zero` element (7) and assign it to measure the distance of `s` from itself (8). Representatives of numbers larger than zero are created by adding successors (9), used in rule (10) to measure the `s`-distance of further vertices reached via `edge`. Note that a vertex might be assigned more than one distance if it is reachable through paths of different lengths. Two details are significant: (i) successors are only created for elements that are already used as distances (9), which ensures that the creation of successors terminates naturally in acyclic graphs; and (ii) establishing the relation of vertices to distances in rule (10) is independent of the creation of new successor elements in (9), which ensures that `succ` forms a unique chain, used globally to measure distances. Together, (i) and (ii) imply that only a linear number of new elements are created in acyclic graphs, even though such graphs might contain an exponential number of distinct paths.

Using the linear order of `succ`, rules (11)–(15) axiomatise arithmetic relationships in the usual way. Relation names have the expected intended meaning: `add`(x, y, z) means "$x + y = z$", `mul`(x, y, z) means "$x * y = z$", and `exp`(x, y) means "$2^x = y$." Finally, rule (16) derives `goal` if an `s`–`t` path of the required length is discovered.

$$\mathtt{edge}(v_1, v_2) \to \mathtt{path}(v_1, v_2) \tag{4}$$

$$\mathtt{edge}(v_1, v_2) \wedge \mathtt{path}(v_2, v_3) \to \mathtt{path}(v_1, v_3) \tag{5}$$

$$\mathtt{path}(v, v) \to \mathtt{goal} \tag{6}$$

$$\to \exists x.\mathtt{zero}(x) \tag{7}$$

$$\mathtt{zero}(x) \to \mathtt{dist}(\mathtt{s}, x) \tag{8}$$

$$\mathtt{dist}(v, x) \to \exists x'.\mathtt{succ}(x, x') \tag{9}$$

$$\mathtt{dist}(v_1, x_1) \wedge \mathtt{edge}(v_1, v_2) \wedge \mathtt{succ}(x_1, x_2) \to \mathtt{dist}(v_2, x_2) \tag{10}$$

$$\mathtt{zero}(x) \wedge \mathtt{dist}(v, y) \to \mathtt{add}(x, y, y) \wedge \mathtt{mul}(x, y, x) \tag{11}$$

$$\mathtt{add}(x, y, z) \wedge \mathtt{succ}(x, x') \wedge \mathtt{succ}(z, z') \to \mathtt{add}(x', y, z') \tag{12}$$

$$\mathtt{mul}(x, y, z) \wedge \mathtt{succ}(x, x') \wedge \mathtt{add}(z, y, z') \to \mathtt{mul}(x', y, z') \tag{13}$$

$$\mathtt{zero}(x) \wedge \mathtt{succ}(x, x') \to \mathtt{exp}(x, x') \tag{14}$$

$$\mathtt{exp}(x, y) \wedge \mathtt{succ}(x, x') \wedge \mathtt{add}(y, y, y') \to \mathtt{exp}(x', y') \tag{15}$$

$$\mathtt{mul}(x, x, y) \wedge \mathtt{exp}(y, y') \wedge \mathtt{exp}(y', z) \wedge \mathtt{dist}(\mathtt{t}, z) \to \mathtt{goal} \tag{16}$$

$$\mathtt{dist}(v, x) \wedge \mathtt{goal} \to \mathtt{succ}(x, x) \tag{17}$$

Figure 1 Rule set in $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ that expresses the DK query, with termination guaranteed after polynomially many chase steps.

As argued above, the rules terminate polynomially on acyclic graphs (even when using the skolem chase). If there are cycles, however, paths can be of unbounded length, leading to a non-terminating creation of successor elements in (9). This is prevented by rule (17), which entails `succ`-loops on all distances once `goal` was derived, thus blocking further applications of (9) in the standard chase. In the Datalog-first chase, cycles will be detected before creating any elements, and `succ`-loops are established before considering (9). Therefore, this chase terminates after polynomially many steps on cyclic graphs as well. ◀

Although the rules of Figure 1 are in $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$, they do not fall into the RMFA fragment of Carral et al. [12]. Indeed, we obtain termination by a case distinction: either the graph is cyclic and existential rules will be blocked, or the graph is acyclic and existential rules will apply at most once for each vertex. It is open how this type of reasoning by cases can be integrated into practical termination criteria.

Also note that the rules of Figure 1 are neither in $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ nor in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. The skolem chase produces an infinite `succ`-chain if an `edge`-cycle is reachable from `s`, and rule (17) cannot prevent the application of rule (9) in this chase. The standard chase does have the ability to block rule (9), but it fails to do so if its strategy is to apply rule (9) before rule (17).

Indeed, our rule (17) effectively acts as an “emergency break” for the chase. Similar devices were considered before. Grahne and Onet introduce so-called *denial constraints*, which are rules that, when applied, stop the chase immediately [23]. Gogacz and Marcinkowski observe that this is also achievable with regular rules in the style of (17), which they call *flooding rules* [22]. This approach is strategy-dependent since it requires that the “break” is triggered eagerly before creating further unnecessary elements.

Can we modify Figure 1 to work for the standard chase? Even with arbitrary strategies, we require fairness, so all rule matches must eventually be satisfied. This is not enough, however, since rule (17) has an unbounded number of matches that could all be satisfied

$$\rightarrow \exists x.\texttt{zero}(x) \tag{21}$$

$$\texttt{zero}(x) \rightarrow \texttt{dist}(\texttt{s}, x) \wedge \texttt{ins}(\texttt{s}, x, x) \wedge \texttt{done}(x) \tag{22}$$

$$\texttt{dist}(v_1, x_1) \wedge \texttt{edge}(v_1, v_2) \wedge \texttt{done}(x_1) \rightarrow \exists x_2.\texttt{ins}(v_2, x_1, x_2) \wedge \texttt{subset}(x_2, x_2) \tag{23}$$

$$\texttt{subset}(x_1, x_2) \wedge \texttt{ins}(v, x_0, x_1) \rightarrow \texttt{ins}(v, x_2, x_2) \wedge \texttt{subset}(x_0, x_2) \tag{24}$$

$$\texttt{subset}(x_1, x_2) \wedge \texttt{zero}(x_1) \rightarrow \texttt{ins}(\texttt{s}, x_2, x_2) \wedge \texttt{done}(x_2) \tag{25}$$

$$\texttt{dist}(v_1, x_1) \wedge \texttt{edge}(v_1, v_2) \wedge \texttt{ins}(v_2, x_1, x_2) \rightarrow \texttt{dist}(v_2, x_2) \wedge \texttt{succ}(x_1, x_2) \tag{26}$$

Figure 2 Rules for creating sets of vertices to represent simple paths.

too late. An all-strategies terminating version can be obtained by modifying (17) to require only a single satisfied match to halt all computation globally. To this end, we introduce two new unary relation names `block` and `real`, remove the rules (9) and (17), add new rules:

$$\rightarrow \exists x.\texttt{block}(x) \wedge \texttt{succ}(x, x) \tag{18}$$

$$\texttt{dist}(v, x) \wedge \texttt{block}(y) \rightarrow \exists x'.\texttt{succ}(x, x') \wedge \texttt{real}(x') \wedge \texttt{succ}(x', y) \tag{19}$$

$$\texttt{block}(x) \wedge \texttt{goal} \rightarrow \texttt{real}(x) \tag{20}$$

and replace every body atom of the form $\texttt{succ}(e, f)$ with a conjunction $\texttt{succ}(e, f) \wedge \texttt{real}(f)$ in any of the other rules.

Intuitively, `block` defines a single element that acts as a universal blocker for all potential applications of rule (19). However, rules are restricted to work with successors that are marked as `real`, which is initially not the case for the blocking element. The block becomes "real" and therefore effective when rule (20) is applied. In contrast to the earlier version, (20) only needs to be applied for a single match. By fairness, any cyclic graph will eventually lead to the derivation of `goal` and hence to the application of (20). Therefore, the modified rule set is in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, but there termination can take arbitrarily long depending on the strategy.

This result connects to recent observations of Gogacz et al., who show that fairness is irrelevant for standard chase termination if all rules have only one atom per rule head [34]. They noted, however, that this no longer holds if rules may have multiple head atoms. Our construction requires such larger heads in all existential rules that should be affected by the global blocker, i.e., rule (19) in our example.

We have therefore learned that the standard chase, too, is strictly more expressive than the skolem chase. We can strengthen this result to obtain an upper bound for the size of the chase, albeit not a polynomial one:

▶ **Theorem 13.** *There is a rule set $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ and BCQ q that realise the DK query. Moreover, the standard chase on Σ is at most exponential in the size of the input database.*

It remains open whether this can be strengthened to obtain a polynomial runtime guarantee – we conjecture that the exponential increase in effort is unavoidable. Indeed, the rule set used to show Theorem 13 may result in a chase of exponential size, even when using the Datalog-first strategy.

Proof of Theorem 13. A bigger change in our original approach is now needed. Instead of constructing a unique `succ`-chain to measure distances, we build a tree-like `succ`-structure that grows one branch for every `s`-path, and which stops to grow when encountering cycles. In other words, every element of the `succ`-structure represents a simple path. To accomplish

$$\texttt{first}(v) \to \exists x.\texttt{start}(x, x, v) \wedge \texttt{end}(x) \tag{27}$$

$$\texttt{start}(x, u, v) \wedge \texttt{end}(u) \wedge \texttt{next}(v, v') \to \exists y_1, y_2.\texttt{start}(y_1, x, v') \wedge \texttt{succ}(y_1, y_2) \wedge \texttt{end}(y_2) \tag{28}$$

$$\texttt{start}(x, u, v) \wedge \texttt{succ}(u, u') \to \exists y.\texttt{left}(x, y) \wedge \texttt{start}(y, u', v) \tag{29}$$

$$\texttt{left}(x, y) \to \exists y'.\texttt{right}(x, y') \wedge \texttt{succ}(y, y') \tag{30}$$

$$\texttt{right}(x, y) \wedge \texttt{succ}(x, x') \to \exists y'.\texttt{left}(x', y') \wedge \texttt{succ}(y, y') \tag{31}$$

$$\texttt{end}(x) \wedge \texttt{right}(x, y) \to \texttt{end}(y) \tag{32}$$

$$\texttt{start}(x, u, v) \wedge \texttt{end}(u) \wedge \texttt{last}(v) \wedge \texttt{succ}(x, x') \to \texttt{chain}(x, x') \tag{33}$$

$$\texttt{chain}(x, x') \wedge \texttt{succ}(x', x'') \to \texttt{chain}(x', x'') \tag{34}$$

Figure 3 Rule set to generate a k-exponentially long chain for the proof of Theorem 14.

this, we associate each element with the set of all vertices that have previously been visited along this path. The rules (7)–(10) and (17) in Figure 1 are replaced by the rules in Figure 2. We use facts of the form $\texttt{ins}(v, x, y)$ to express that y is the result of inserting v into the set x, i.e., "$\{v\} \cup x = y$." In particular, $\texttt{ins}(v, x, x)$ can be read as "$v \in x$." Rules (21) and (22) initialise a zero element to encode $\{\texttt{s}\}$. Rule (23) creates a new vertex set when required for extending a path, where $\texttt{done}(x_1)$ asserts that set x_1 was fully initialised and $\texttt{subset}(x_1, x_2)$ states that x_2 contains all elements of x_1. Rules (24) and (25) propagate $\texttt{subset}$ to farther ancestors while establishing that all previously added vertices are also in the newly created set. A set is only considered $\texttt{done}$ when when reaching the zero element $\{\texttt{s}\}$ (25). Finally, rule (26) uses the vertex sets to measure distances. The resulting $\texttt{succ}$ branches are used as a basis for arithmetic operations as before, using rules (11)–(16).

At any given intermediate stage of the chase, we can associate any element c with a set $[c]$ that contains all elements e for which there is a fact $\texttt{ins}(e, c, c)$. Clearly, the unique zero element c_0 with $\texttt{zero}(c_0)$ satisfies $[c_0] = \{\texttt{s}\}$. By an easy induction, one can show that elements c_n for which $\texttt{done}(c_n)$ was derived satisfy the following: there is a unique chain of facts $\texttt{ins}(e_1, c_0, c_1), \ldots, \texttt{ins}(e_n, c_{n-1}, c_n)$ with $c_i \neq c_{i+1}$, and we have $[c_n] = \{\texttt{s}, e_1, \ldots e_n\}$ and $\texttt{subset}(c_i, c_n)$ for all $0 \leq i \leq n$. The existence of these facts implies that a match h of rule (23) is always satisfied if $h(v) \in [h(x_1)]$, i.e., the rule can only be used to create strictly larger sets. Since the size of any set of vertices is bounded by the size of the input, there are at most exponentially many such elements, even if the graph has cycles.

If the graph is acyclic, then the $\texttt{succ}$ relation forms a tree structure that corresponds to an unravelling of the directed acyclic graph rooted in $\texttt{s}$. The distance-related checks work as in Theorem 12. If the graph is cyclic, then $\texttt{succ}$ might also contain cycles, but is still finite. Rules (4)–(6) will detect the cycle and lead to acceptance, as required. ◀

7 Beyond Polynomial Time

After observing the superior expressive power of the standard and Datalog-first chase on polynomial time problems, we turn to the question of whether one can also express queries of higher complexity in rule sets that are guaranteed to terminate in these chases. The answer is a resounding *yes*:

▶ **Theorem 14.** *There is a rule set* $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ *and a BCQ* q *that express a non-elementary Boolean query.*

Proof. We reduce from the following non-elementary decision problem:

Input:	A Turing machine $\mathcal{M}$ and a number k
Question:	When started on the empty tape, does $\mathcal{M}$ halt in at most $\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{k \text{ times}}$ steps?

The number k is encoded by input facts $\mathtt{first}(e_0), \mathtt{next}(e_0, e_1), \ldots, \mathtt{next}(e_{k-1}, e_k), \mathtt{last}(e_k)$. We use the chase to construct a chain of the required k-exponential length. A simulator for k-exponential Turing machine computations can then be implemented with Datalog rules using a standard construction [16]. Note that the query result on inputs that do not use the required encoding (of k or the Turing machine) is irrelevant for hardness; yet we must ensure that the chase terminates on such inputs.

We first describe the basic construction of the k-exponential chain and discuss termination later. The rules in Figure 3 produce a series of k full binary trees of depth $1, 2, 2^2, 2^{2^2}, \ldots$ Each tree starts on the second level (containing two elements) and uses relation names `left` and `right` to define a node's children. Nodes on the same level form a chain $\mathtt{succ}(n_1, n_2), \mathtt{succ}(n_2, n_3), \ldots, \mathtt{succ}(n_{\ell-1}, n_\ell)$; the first element n_1 is marked by a fact $\mathtt{start}(n_1, u, v)$ where u defines the level of the tree, and v defines the number of the tree; the last element n_ℓ is marked by $\mathtt{end}(n_\ell)$.

Rule (27) creates a one-element "tree" as a start (not in the above list of k rootless trees). Rule (28) initialises the first level of the next tree, using the `succ`-chain of the last level of the previous tree to count up the levels. Subsequent levels of the tree are initialised by (29) and completed by rules (30) and (31). The next level's last element is marked by (32). Rules (33) and (34) define the last level of the last tree as the required chain.

It is not hard to see that the rules in Figure 3 terminate, even in the skolem chase, if the `next`-graph does not have cycles and the database only mentions the relation names `first`, `next`, and `last`. To ensure termination in case of `next`-cycles, we can add cycle-detection rules similar to (4)–(6). To ensure termination on all database instances (i.e., such that already contain facts using other relation names such as `succ` or `start`), we need to detect similar cyclic arrangements involving `succ` or `start`. Termination of the Datalog-first chase can then be ensured by adding "flooding rules" akin to (17), in this case creating ubiquitous `start`, `left`, `right`, and `succ` relations, so as to block rules (28)–(31). ◀

As discussed in Section 6, the flooding rules used to ensure termination of the Datalog-first chase can be replaced by introducing a global blocking element that can be activated in a single rule application when a cycle is detected. Fairness then also ensures the termination of the standard chase, and we obtain:

▶ **Theorem 15.** *There is a rule set* $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ *and a BCQ* q *that express a non-elementary Boolean query.*

It is not difficult to define a fixed `next`-chain in rules, so as to use a constant tower of k exponentials instead of a data-dependent one. This yields rule sets that realise k-ExpTime-complete queries, which by the Time Hierarchy theorems cannot be realised in $(k-1)$-exponential time. This yields the following:

▶ **Theorem 16.** *The classes of rule sets in* $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ *that terminate after at most* k*-exponentially many steps form a hierarchy of strictly increasing expressivity. The same applies to* k*-exponentially terminating rule sets in* $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$.

8 Conclusion

We have studied classes $\mathsf{CT}^{\mathsf{x}}_{\forall(\forall)}$ of existential rules for which a certain chase variant $\mathsf{x} \in \{\mathsf{sk}, \mathsf{std}, \mathsf{dlf}, \mathsf{core}\}$ terminates on all database instances, and (where applicable) under all strategies. To review our results, it is meaningful to further distinguish chase termination classes by upper bounds in terms of the size of the input database. For example, $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\text{poly})$ would denote the subset of $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ where chase termination is guaranteed after polynomial time. Now given such a class CT, we investigated the set $\llbracket\mathsf{CT}\rrbracket$ of all abstract queries (sets of concrete databases over an input signature) that can be expressed by some theory from CT. Using this notation, our main results are as follows:

$$\begin{array}{ccccccccccccc}
\llbracket\mathsf{Datalog}\rrbracket & \overset{\text{(Thm 9)}}{=} & \llbracket\mathsf{CT}^{\mathsf{sk}}_{\forall}\rrbracket & \overset{\text{(Thm 12)}}{\subset} & \llbracket\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\text{poly})\rrbracket & \overset{\text{(Thm 16)}}{\subset} & \llbracket\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\text{exp})\rrbracket & \overset{\text{(Thm 16)}}{\subset} & \ldots & \subset & \llbracket\bigcup_k \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(k\text{-exp})\rrbracket & \overset{\text{(Thm 14)}}{\subset} & \llbracket\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}\rrbracket \\
 & & {\scriptstyle\text{(Thm 8)}}\ \Vert & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} & & & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\
 & & \llbracket\mathsf{CT}^{\mathsf{sk}}_{\forall}(\text{poly})\rrbracket & \subseteq & \llbracket\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\text{poly})\rrbracket & \overset{\text{(Thm 16)}}{\subset} & \llbracket\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\text{exp})\rrbracket & \overset{\text{(Thm 16)}}{\subset} & \ldots & \subset & \llbracket\bigcup_k \mathsf{CT}^{\mathsf{std}}_{\forall\forall}(k\text{-exp})\rrbracket & \overset{\text{(Thm 15)}}{\subset} & \llbracket\mathsf{CT}^{\mathsf{std}}_{\forall\forall}\rrbracket
\end{array}$$

Many further questions remain open, and indicate promising directions for future research:

Absolute expressibility. A terminating chase can only express queries that are decidable and closed under homomorphism, but we saw that the skolem chase can express much less. Some of the other chase variants might actually capture this class of queries. Even if not, it would be interesting to characterise their expressivity semantically.

Relative expressibility. We know that $\llbracket\mathsf{CT}^{\mathsf{std}}_{\forall\forall}\rrbracket \subseteq \llbracket\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}\rrbracket \subseteq \llbracket\mathsf{CT}^{\mathsf{core}}_{\forall}\rrbracket$, but it remains open if any of these inclusions are strict. If some are equalities, it would be interesting to find computable rewritings that produce rule sets for which a weaker chase terminates.

Complexity relationships. Theorem 13 achieved termination for the standard chase at the cost of an exponential runtime increase. We conjecture that this is unavoidable, and that $\llbracket\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\text{poly})\rrbracket \subset \llbracket\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\text{poly})\rrbracket$. Can all queries in $\llbracket\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}\rrbracket$ be implemented in worst-case optimal time? And can the standard chase express the same queries at an exponential penalty? Do we even have $\llbracket\mathsf{CT}^{\mathsf{sk}}_{\forall}\rrbracket \neq \llbracket\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\text{poly})\rrbracket$?

Decidable termination criteria. None of our beyond-skolem queries satisfy any of the known termination criteria. New approaches are needed to encompass our examples.

Termination on restricted database classes. We required termination on all databases, using databases with restricted EDB signatures only to define query realisation. Requiring termination only on databases over EDB relations might lead to larger classes of rule sets, possibly with higher expressivity. This certainly occurs when restricting to specific "well-formed" instance databases: if we would exclude cyclic databases, even the skolem chase could express non-elementary queries. However, such restrictions are also enough to capture all of PTime (assuming negation and order to be axiomatised), and even unrealistically powerful classes, such as non-uniform $\text{P}_{/\text{poly}}$.

Practical applications. Practical implementations for standard chase and also for Datalog-first chase exist, so it is promising to explore the use of beyond-skolem expressive power in applications. Specific uses could help guide the theoretical research.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley, 1994.

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

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

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

5 Bruce L. Bauslaugh. Core-like properties of infinite graphs and structures. *Discrete Math.*, 138(1):101–111, 1995.

6 Catriel Beeri and Moshe Y. Vardi. The Implication Problem for Data Dependencies. In Shimon Even and Oded Kariv, editors, *Proc. 8th Colloquium on Automata, Languages and Programming (ICALP'81)*, volume 115 of *LNCS*, pages 73–85. Springer, 1981.

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

8 Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the Chase. In *Proc. 36th Symposium on Principles of Database Systems (PODS'17)*, pages 37–52. ACM, 2017.

9 Angela Bonifati, Ioana Ileana, and Michele Linardi. Functional Dependencies Unleashed for Scalable Data Exchange. In Peter Baumann, Ioana Manolescu-Goujot, Luca Trani, Yannis E. Ioannidis, Gergely Gábor Barnaföldi, László Dobos, and Evelin Bányai, editors, *Proc. 28th Int. Conf. on Scientific and Statistical Database Management (SSDBM'16)*, pages 2:1–2:12. ACM, 2016.

10 Marco Calautti and Andreas Pieris. Oblivious Chase Termination: The Sticky Case. In *Proc. 22nd Int. Conf. on Database Theory (ICDT'19)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019.

11 Andrea Calì, Georg Gottlob, and Andreas Pieris. Query Answering under Non-guarded Rules in Datalog+/-. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Proc. 4th Int. Conf. on Web Reasoning and Rule Systems (RR 2010)*, volume 6333 of *LNCS*, pages 1–17. Springer, 2010.

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

13 David Carral, Markus Krötzsch, Maximilian Marx, Ana Ozaki, and Sebastian Rudolph. Preserving Constraints with the Stable Chase. In Benny Kimelfeld and Yael Amsterdamer, editors, *Proc. 21st Int. Conf. on Database Theory (ICDT'18)*, volume 98 of *LIPIcs*, pages 12:1–12:19. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018.

14 Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases.* Springer, 1990.

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

16 Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

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

18 Stathis Delivorias, Michel Leclère, Marie-Laure Mugnier, and Federico Ulliana. On the k-Boundedness for Existential Rules. In Christoph Benzmüller, Francesco Ricca, Xavier Parent, and Dumitru Roman, editors, *Proc. 2nd Int. Joint Conf. on Rules and Reasoning (RuleML+RR'18)*, volume 11092 of *LNCS*, pages 48–64. Springer, 2018.

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

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

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

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

23 Gösta Grahne and Adrian Onet. Anatomy of the Chase. *Fundam. Inform.*, 157(3):221–270, 2018.

24 Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discrete Math.*, 109:117–126, 1992.

25 Markus Krötzsch and Sebastian Rudolph. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In Toby Walsh, editor, *Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11)*, pages 963–968. AAAI Press/IJCAI, 2011.

26 David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4:455–469, 1979.

27 Bruno Marnette. Generalized Schema-Mappings: from Termination to Tractability. In Jan Paredaens and Jianwen Su, editors, *Proc. 28th Symposium on Principles of Database Systems (PODS'09)*, pages 13–22. ACM, 2009.

28 Michael Meier, Michael Schmidt, and Georg Lausen. On Chase Termination Beyond Stratification. *PVLDB*, 2(1):970–981, 2009.

29 Michaël Thomazo Michel Leclére, Marie-Laure Mugnier and Federico Ulliana. A Single Approach to Decide Chase Termination on Linear Existential Rules. *CoRR*, abs/1810.02132, 2018. arXiv:1810.02132.

30 Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In *Proc. 28th AAAI Conf. on Artif. Intell. (AAAI'14)*, pages 129–137. AAAI Press, 2014.

31 Sebastian Rudolph. The Two Views on Ontological Query Answering. In Georg Gottlob and Jorge Pérez, editors, *Proc. 8th Alberto Mendelzon Workshop on Foundations of Data Management (AMW'14)*, volume 1189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

32 Sebastian Rudolph and Michaël Thomazo. Characterization of the Expressivity of Existential Rule Queries. In Qiang Yang and Michael Wooldridge, editors, *Proc. 24th Int. Joint Conf. on Artificial Intelligence (IJCAI'15)*, pages 3193–3199. AAAI Press, 2015.

33 Sebastian Rudolph and Michaël Thomazo. Expressivity of Datalog Variants - Completing the Picture. In Subbarao Kambhampati, editor, *Proc. 25th Int. Joint Conf. on Artificial Intelligence (IJCAI'15)*, pages 1230–1236. AAAI Press, 2016.

34 Andreas Pieris Tomasz Gogacz, Jerzy Marcinkowski. All-Instances Restricted Chase Termination: The Guarded Case. *CoRR*, abs/1901.03897, 2019. arXiv:1901.03897.

35 Jacopo Urbani, Markus Krötzsch, Ceriel J. H. Jacobs, Irina Dragoste, and David Carral. Efficient Model Construction for Horn Logic with VLog: System description. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Proc. 9th Int. Joint Conf. on Automated Reasoning (IJCAR'18)*, volume 10900 of *LNCS*, pages 680–688. Springer, 2018.

36 Heng Zhang, Yan Zhang, and Jia-Huai You. Existential Rule Languages with Finite Chase: Complexity and Expressiveness. In Blai Bonet and Sven Koenig, editors, *Proc. 29th AAAI Conf. on Artificial Intelligence (AAAI'15)*. AAAI Press, 2015.