Stop the Chase: Short Contribution

Michael Meier*, Michael Schmidt* and Georg Lausen

University of Freiburg, Institute for Computer Science
Georges-Köhler-Allee, 79110 Freiburg, Germany
{meierm,mschmidt,lausen}@informatik.uni-freiburg.de

Abstract. The chase procedure, an algorithm proposed 25+ years ago to fix constraint violations in database instances, has been successfully applied in a variety of contexts, such as query optimization and data exchange. Its practicability, however, is limited by the fact that – for an arbitrary set of constraints – it might not terminate; even worse, chase termination is an undecidable problem in general. In response, the database community has proposed sufficient restrictions on top of the constraints that guarantee chase termination on any database instance. In this paper, we propose a sufficient termination condition, called *inductive restriction*, which strictly generalizes previous conditions, but can be checked as efficiently.

1 Introduction

The chase procedure is a fundamental algorithm that has been successfully applied in a variety of database applications [7, 10, 5, 9, 11, 15, 2, 1, 13]. Originally proposed to tackle the implication problem for data dependencies [7, 5] and to optimize Conjunctive Queries (CQs) under data dependencies [3, 10], it has become a central tool in Semantic Query Optimization (SQO) [14, 1, 16]. For instance, the chase can be used to enumerate minimal CQs under a set of dependencies [1], thus supporting the search for more efficient query evaluation plans. Beyond SQO, it has been applied in many other contexts, such as data exchange [15], peer data exchange [2], data integration [11], query answering using views [9], and probabilistic databases [13].

The core idea of the chase algorithm is simple: given a set of dependencies (also called constraints) over a database schema and an instance as input, it fixes constraint violations in the instance. One problem with the chase, however, is that – given an arbitrary set of constraints – it might never terminate; even worse, this problem is undecidable in general, also for a fixed instance [4]. Addressing this issue, sufficient conditions for the constraints that guarantee termination on any database instance have been proposed [15, 4, 16]. Such conditions are the central topic in this paper. We introduce the class of *inductively restricted constraints*, for which the chase terminates in polynomial time data complexity. Like existent sufficient termination conditions, inductive restriction asserts that there are no positions in the schema where fresh labeled nulls might be cyclically created during chase application. It relies on a sophisticated study of (a) positions in the database schema where null values might appear, (b) subsets of the constraints that cyclically pass null values, and (c) connections between such cycles. The combination

* The work of this author was funded by DFG grant GRK 806/3.

of these aspects makes inductive restriction more general than previous sufficient termination conditions, thus making a larger class of constraints amenable to the chase.
Structure. We start with some preliminaries in the following section. Section 3 introduces inductive restriction, our sufficient data-independent termination condition. Finally, Section 4 concludes the paper.
Remark. An extended version of this paper including full proofs can be found in [12].

2 Preliminaries

General mathematical notation. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, ..., n\}$. For a set M, we denote by 2^M its powerset.
Databases. We fix three pairwise disjoint infinite sets: the set of *constants* Δ, the set of *labeled nulls* Δ_{null}, and the set of *variables* V. A *database schema* $\mathcal{R}$ is a finite set of relational symbols $\{R_1, ..., R_n\}$. In the rest of the paper, we assume the database schema and the set of constants and labeled nulls to be fixed. A *database instance* I is a finite set of $\mathcal{R}$-atoms that contains only elements from $\Delta \cup \Delta_{null}$ in its positions. We denote an element of an instance as *fact*. The domain of I, $dom(I)$, is the set of elements from $\Delta \cup \Delta_{null}$ that appear in I.
We use the term *position* to denote a position in a predicate, e.g. a three-ary predicate R has three positions R^1, R^2, R^3. We say that a variable, labeled null, or constant c appears e.g. in a position R^1 if there exists a fact $R(c, ...)$.
Constraints. Let $\overline{x}, \overline{y}$ be tuples of variables. We consider two types of database constraints: *tuple generating dependencies* (TGDs) and *equality generating dependencies* (EGDs). A TGD has the form $\alpha := \forall \overline{x}(\phi(\overline{x}) \rightarrow \exists \overline{y}\psi(\overline{x}, \overline{y}))$ such that both ϕ and ψ are conjunctions of atomic and equality-free $\mathcal{R}$-atoms, possibly with parameters from Δ and all variables from $\overline{x}$ that occur in ψ must also occur in ϕ. We denote by $pos(\alpha)$ the set of positions in ϕ. An EGD has the form $\alpha := \forall \overline{x}(\phi(\overline{x}) \rightarrow x_i = x_j)$, where x_i, x_j occur in ϕ and ϕ is a non-empty conjunction of equality-free $\mathcal{R}$-atoms, possibly with parameters from Δ. We denote by $pos(\alpha)$ the set of positions in ϕ. As a notational convenience, we will often omit the $\forall$-quantifier and respective list of universally quantified variables. For a set of TGDs and EGDs Σ we set $pos(\Sigma) := \bigcup_{\xi \in \Sigma} pos(\xi)$.
Chase. We assume that the reader is familiar with the chase procedure and give only a short introduction here, referring the interested reader to [15] for a more detailed discussion. A chase step $I \overset{\alpha,\overline{a}}{\rightarrow} J$ takes a relational database instance I such that $I \nvDash \alpha(\overline{a})$ and adds tuples (in case of TGDs) or collapses some elements (in case of EGDs) such that the resulting relational database J is a model of $\alpha(\overline{a})$. If J was obtained from I in that kind, we sometimes also write $I\overline{a} \oplus C_\alpha$ instead of J. A chase sequence is an exhaustive application of applicable constraints $I_0 \overset{\alpha_0,\overline{a}_0}{\longrightarrow} I_1 \overset{\alpha_1,\overline{a}_1}{\longrightarrow} \ldots$, where we impose no strict order on what constraint to apply in case several constraints are applicable. If this sequence is finite, say I_r being its final element, the chase terminates and its result I_0^Σ is defined as I_r. The length of this chase sequence is r. Note that different orders of application orders may lead to a different chase result. However, as proven in [15], two different chase orders always lead to homomorphically equivalent results, if these exist. Therefore, we write I^Σ for the result of the chase on an instance I under constraints Σ. It has been shown in [7, 5, 10] that $I^\Sigma \models \Sigma$. If a chase step cannot be performed (e.g.,

because application of an EGD would have to equate two constants) or in case of an infinite chase sequence, the result of the chase is undefined.

3 Data-independent Chase Termination

In the past, sufficient conditions for constraint sets have been developed that guarantee chase termination for any instance. One such condition is *weak acyclicity* [15], which asserts that there are no cyclically connected positions in the constraint set that may introduce fresh labeled null values, by a global study of relations between the constraints. In [4], weak acyclicity was generalized to *stratification*, which enforces weak acyclicity only locally, for subsets of constraints that might cyclically cause to fire each other. We further generalized stratification to *safe restriction* in [16]. We start by reviewing its central ideas and formal definition, which form the basis for our novel condition *inductive restriction*.

Safe Restriction. The idea of safe restriction is to keep track of positions where fresh null values might be created in or copied to. As a basic tool, we borrow the definition of *affected positions* from [6]. We emphasize that, in [6], this definition has been used in a different context: there, the constraints are interpreted as axioms that are used to derive new facts from the database and the problem is query answering on the implied database, using the chase as a central tool.

Definition 1. *[6]* Let Σ be a set of TGDs. The set of *affected positions* $\text{aff}(\Sigma)$ is defined inductively as follows. Let π be a position in the head of an $\alpha \in \Sigma$.

- If an existentially quantified variable appears in π, then $\pi \in \text{aff}(\Sigma)$.
- If the same universally quantified variable X appears both in position π, and only in affected positions in the body of α, then $\pi \in \text{aff}(\Sigma)$. □

Akin to the dependency graph in weak acyclicity [15], we define a safety condition that asserts the absence of cycles through constraints that may introduce fresh null values. As an improvement, we exhibit the observation that only values created due to or copied from affected positions may cause non-termination. We introduce the notion of *propagation graph*, which refines the dependency graph from [15] by taking affected positions into consideration.

Definition 2. Let Σ be a set of TGDs. We define a directed graph called *propagation graph* $\text{prop}(\Sigma) := (\text{aff}(\Sigma), E)$ as follows. There are two kinds of edges in E. Add them as follows: for every TGD $\forall\overline{x}(\phi(\overline{x}) \rightarrow \exists\overline{y}\psi(\overline{x},\overline{y})) \in \Sigma$ and for every x in $\overline{x}$ that occurs in ψ and every occurrence of x in ϕ in position π_1

- if x occurs only in affected positions in ϕ then, for every occurrence of x in ψ in position π_2, add an edge $\pi_1 \rightarrow \pi_2$ (if it does not already exist).
- if x occurs only in affected positions in ϕ then, for every existentially quantified variable y and for every occurrence of y in a position π_2, add a special edge $\pi_1 \stackrel{*}{\rightarrow} \pi_2$ (if it does not already exist). □

Definition 3. A set Σ of constraints is called *safe* iff $\text{prop}(\Sigma)$ has no cycles going through a special edge. □

Safety is a sufficient termination condition which strictly generalizes weak acyclicity and is different from stratification [16]. The idea behind safe restriction now is to assert safety locally, for subsets of the constraints that may cyclically cause each other to fire in such a way that null values are passed in these cycles.

Definition 4. Let Σ abe given and $P \subseteq pos(\Sigma)$. For all $\alpha, \beta \in \Sigma$, we define $\alpha \prec_P \beta$ iff there are tuples $\overline{a}, \overline{b}$ and a database instance I s.t. (i) $I \not\models \alpha(\overline{a})$, (ii) $I \models \beta(\overline{b})$, (iii) $I \overset{\alpha,\overline{a}}{\rightarrow} J$, (iv) $J \not\models \beta(\overline{b})$, (v) I contains null values only in positions from P and (vi) there is a null value $n \in \overline{b} \cap \Delta_{null}$ in the head of $\beta(\overline{b})$. □

Informally, $\alpha \prec_P \beta$ holds if α might cause β to fire s.t., when null values occur only in positions from P, β copies some null values. We next introduce a notion for affected positions relative to a constraint and a set of positions.

Definition 5. For any set of positions P and a TGD α let aff-cl(α, P) be the set of positions π from the head of α such that

- for every universally quantified variable x in π: x occurs in the body of α only in positions from P or
- π contains an existentially quantified variable. □

On top of previous definitions we introduce the central tool of *restriction systems*.

Definition 6. A *restriction system* is a pair $(G'(\Sigma), f)$, where $G'(\Sigma) := (\Sigma, E)$ is a directed graph and $f : \Sigma \rightarrow 2^{pos(\Sigma)}$ is a function such that

- forall TGDs α and forall $(\alpha, \beta) \in E$: aff-cl$(\alpha, f(\alpha)) \cap pos(\{\beta\}) \subseteq f(\beta)$,
- forall EGDs α and forall $(\alpha, \beta) \in E$: $f(\alpha) \cap pos(\{\beta\}) \subseteq f(\beta)$, and
- forall $\alpha, \beta \in \Sigma$: $\alpha \prec_{f(\alpha)} \beta \implies (\alpha, \beta) \in E$.

A restriction system is *minimal* if it is obtained from $((\Sigma, \emptyset), \{(\alpha, \emptyset) \mid \alpha \in \Sigma\})$ by a repeated application of the constraints from bullets one to three (until all constraints hold) s.t., in case of the first and second bullet, the image of $f(\beta)$ is extended only by those positions that are required to satisfy the condition. □

Example 1. Let predicate E(x,y) store graph edges and predicate S(x) store some nodes. The constraints $\Sigma = \{\alpha_1, \alpha_2\}$ with $\alpha_1 :=$ S(x), E(x,y) $\rightarrow$ E(y,x) and $\alpha_2 :=$ S(x), E(x,y) $\rightarrow \exists z$ E(y,z), E(z,x) assert that all nodes in S have a cycle of length 1 and 2. It holds that aff(Σ) = $\{\text{E}^1,\text{E}^2\}$ and it is easy to verify that Σ is neither safe nor stratified (see Def. 2 in [4]). The minimal restriction system for Σ is G'(Σ):=(Σ,$\{(\alpha_2,\alpha_1)\}$) with f(α_1) := $\{\text{E}^1,\text{E}^2\}$ and f(α_2) := $\emptyset$; in particular, $\alpha_1 \not\prec_{f(\alpha_1)} \alpha_1$, $\alpha_1 \not\prec_{f(\alpha_1)} \alpha_2$, $\alpha_2 \prec_{f(\alpha_2)} \alpha_1$, and $\alpha_2 \not\prec_{f(\alpha_2)} \alpha_2$ hold. □

As shown in [16], the minimal restriction system is unique and can be computed by an NP-algorithm. We are ready to define the notion of safe restriction:

Definition 7. Σ is called *safely restricted* if and only if every strongly connected component of its minimal restriction system is safe. □

Example 2. Constraint set Σ from Example 1 is safely restricted: its minimal restriction system contains no strongly connected components. □

part(Σ: Set of TDGs and EGDs) {
1: compute the strongly connected components (as sets of constraints) $C_1, \ldots, C_n$ of the minimal restriction system of Σ;
2: $D \leftarrow \emptyset$
3: **if** (n == 1) **then**
4: **if** ($C_1 \neq \Sigma$) **then** return **part**(C_1); **endif**
7: return $\{\Sigma\}$;
8: **endif**
6: **for** i=1 to n **do** $D \leftarrow D \cup$ **part**(C_i); **endfor**
11: return D; }

Fig. 1. Algorithm to compute subsets of Σ.

As shown in [16], safe restriction (a) guarantees chase termination in polynomial time data complexity, (b) is strictly more general than stratification, and (c) it can be checked by a coNP-algorithm if a set of constraints is safely restricted.

Inductive Restriction. We now introduce the novel class of *inductively restricted constraints*, which generalizes safe restriction but, like the latter, gives polynomial-time termination guarantees. We start with a motivating example.

Example 3. We extend the constraints from Example 1 to $\Sigma' := \Sigma \cup \{\alpha_3\}$, where $\alpha_3 := \exists x, y S(x), E(x, y)$. Then G'($\Sigma'$):=($\Sigma'$,{($\alpha_1, \alpha_2$),($\alpha_2$,$\alpha_1$),($\alpha_3$,$\alpha_1$),($\alpha_3$,$\alpha_2$)}) with f($\alpha_1$) = f($\alpha_2$) := {$E^1$,$E^2$,$S^1$} and f($\alpha_3$) := $\emptyset$ is the minimal restriction system. It contains the strongly connected component {α_1,α_2}, which is not safe. Consequently, Σ' is not safely restricted. □

Intuitively, safe restriction does not apply in the example above because α_3 "infects" position S^1 in the restriction system. Though, null values cannot be repeatedly created in S^1: α_3 fires at most once, so it does not affect chase termination. Our novel termination condition recognizes such situations by recursively computing the minimal restriction systems of the strongly connected components. We formalize this computation in Algorithm 1, called *part*(Σ). Based on this algorithm, we define an improved sufficient termination condition.

Definition 8. Let Σ be a set of constraints. We call Σ *inductively restricted* iff for all $\Sigma' \in part(\Sigma)$ it holds that Σ' is safe. □

As stated in the following lemma, inductive restriction strictly generalizes safe restriction, but does not increase the complexity of the recognition problem.

Lemma 1. Let Σ be a set of constraints.

- If Σ is safely restricted, then it is inductively restricted.
- There is some Σ that is inductively restricted, but not safely restricted.
- The recognition problem for inductive restriction is in coNP. □

Example 4. Consider Σ' from Example 3. It is easy to verify that $part(\Sigma') = \emptyset$ and we conclude that Σ' is inductively restricted. As argued in Example 3, Σ' is not safely restricted, which proves the second claim in Lemma 1. □

The next theorem gives the main result of this section, showing that inductive restriction guarantees chase termination in polynomial time data complexity. To the best of our knowledge inductive restriction is the most general sufficient termination condition for the chase that has been proposed so far.

Theorem 1. Let Σ be a fixed set of inductively restricted constraints. Then, there exists a polynomial $Q \in \mathbb{N}[X]$ such that for any database instance I, the length of every chase sequence is bounded by $Q(||I||)$, where $||I||$ is the number of distinct values in I. □

4 Conclusions

We considered the termination of the chase algorithm. As our main contribution, we generalized all sufficient data-independent termination conditions that were known so far. Our results on chase termination directly carry over to applications that rely on the chase and also to the so-called core-chase presented in [4]. There are some interesting open questions left. First, it is unknown if the recognition problem for inductive restriction, which was shown to be in coNP, is also coNP-hard. Second, it is left open if the positive results on core computation in data exchange settings from [8] extend to inductive restriction.

References

1. A. Deutsch et al. Query Reformulation with Constraints. *SIGMOD Record*, 35(1):65–73, 2006.
2. A. Fuxman et al. Peer data exchange. *ACM Trans. Database Syst.*, 31(4):1454–1498, 2006.
3. A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient Optimization of a Class of Relational Expressions. *ACM Trans. Database Syst.*, 4(4):435–454, 1979.
4. Alin Deutsch et al. The Chase Revisited. In *PODS*, pages 149–158, 2008.
5. C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *J. ACM*, 31(4):718–741, 1984.
6. A. Calì, G. Gottlob, and M. Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In *Descr. Logics*, volume 353, 2008.
7. D. Maier et al. Testing Implications of Data Dependencies. In *SIGMOD*, pages 152–152, 1979.
8. G. Gottlob and A. Nash. Efficient Core Computation in Data Exchange. *J. ACM*, 55(2), 2008.
9. A. Y. Halevy. Answering Queries Using Views: A Survey. *VLDB J.*, pages 270–294, 2001.
10. D. S. Johnson and A. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. In *PODS*, pages 164–169, 1982.
11. M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.
12. M. Meier, M. Schmidt, and G. Lausen. Stop the Chase, Technical Report. *CoRR*, abs/0901.3984, 2009.
13. D. Olteanu, J. Huang, and C. Koch. SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases. In *ICDE*, 2009. To appear.
14. L. Popa and V. Tannen. An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. In *ICDT*, pages 39–57, 1999.
15. R. Fagin et al. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
16. M. Schmidt, M. Meier, and G. Lausen. Foundations of SPARQL Query Optimization, Technical Report. *CoRR*, abs/0812.3788, 2008.