On Chase Termination Beyond Stratification

Michael Meier* Michael Schmidt* Georg Lausen
University of Freiburg
Institute for Computer Science
Georges-Köhler-Allee, Building 051
79110 Freiburg i. Br., Germany
{meierm, mschmidt, lausen}@informatik.uni-freiburg.de

ABSTRACT

We study the termination problem of the chase algorithm, a central tool in various database problems such as the constraint implication problem, Conjunctive Query optimization, rewriting queries using views, data exchange, and data integration. The basic idea of the chase is, given a database instance and a set of constraints as input, to fix constraint violations in the database instance. It is well-known that, for an arbitrary set of constraints, the chase does not necessarily terminate (in general, it is even undecidable if it does or not). Addressing this issue, we review the limitations of existing sufficient termination conditions for the chase and develop new techniques that allow us to establish weaker sufficient conditions. In particular, we introduce two novel termination conditions called *safety* and *inductive restriction*, and use them to define the so-called T*-hierarchy* of termination conditions. We then study the interrelations of our termination conditions with previous conditions and the complexity of checking our conditions. This analysis leads to an algorithm that checks membership in a level of the T-hierarchy and accounts for the complexity of termination conditions. As another contribution, we study the problem of *data-dependent* chase termination and present sufficient termination conditions w.r.t. fixed instances. They might guarantee termination although the chase does not terminate in the general case. As an application of our techniques beyond those already mentioned, we transfer our results into the field of query answering over knowledge bases where the chase on the underlying database may not terminate, making existing algorithms applicable to broader classes of constraints.

1. INTRODUCTION

The chase procedure is a fundamental algorithm that has been successfully applied in a variety of database applications [8, 13, 4, 12, 15, 21, 2, 1, 19]. Originally proposed to tackle the implication problem for data dependencies [8, 4] and to optimize Conjunctive Queries (CQs) under data dependencies [3, 13], it has become a central tool in Semantic Query Optimization (SQO) [20, 1, 22]. 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, the chase algorithm has been applied in many other contexts, such as data exchange [21], peer data exchange [2], data integration [15], query answering using views [12], and probabilistic databases [19].

The core idea of the chase algorithm is simple: given a set of dependencies (also called constraints) over a database schema and a fixed database instance as input, it fixes constraint violations in the instance. As a minimal and intuitive scenario we consider a database graph schema that provides a relation $E(\textit{src}, \textit{dst})$, which stores directed edges from node *src* to node *dst*, and a node relation $S(n)$ containing nodes with some distinguished properties, which are enforced by constraints. These constraints will vary from example to example and we denote nodes in S as *special nodes* in the following. We sketch the idea of the chase algorithm using a single constraint $\alpha_1 := \forall x(S(x) \rightarrow \exists y E(x, y))$, stating that each special node has at least one outgoing edge. Now consider the sample database instance $I := \{S(n_1), S(n_2), E(n_1, n_2)\}$. It is easy to see that I does not satisfy α_1, because it does not contain an outgoing edge for special node n_2. In its effort to fix the constraint violations in the database instance, the chase procedure would create the tuple $t_1 := E(n_2, x_1)$, where x_1 is a fresh null value. The resulting database instance $I' := I \cup \{t_1\}$ now satisfies constraint α_1, so the chase terminates and returns I' as result.

One major problem with the chase algorithm, however, is that it does not terminate in the general case. To give an idea of the problem, let us sketch a scenario that induces a non-terminating chase sequence. We replace the constraint α_1 from before by constraint $\alpha_2 := \forall x(S(x) \rightarrow \exists y E(x, y), S(y))$, which asserts that each special node links to another special node. Now consider the instance I from before. Obviously, I does not satisfy α_2, because special node n_2 has no outgoing edge. In response, the chase fixes this constraints violation by adding the two tuples $E(n_2, x_1)$ and $S(x_1)$ to I, where x_1 is a fresh null value. Constraint α_2 is then fixed w.r.t. value n_2, but now the special node x_1 introduced in the last chase step violates α_2. In subsequent steps the chase would add $E(x_1, x_2)$, $S(x_2)$, $E(x_2, x_3)$, $S(x_3)$, ..., where x_2, x_3, ... are fresh null values. Hence, when given instance I and constraint α_2 as input, the chase procedure will never terminate.

As shown in [9], in general it is undecidable if the chase terminates or not, even for a fixed instance. Still, addressing the issue of non-terminating chase sequences, several *sufficient* conditions for the input constraints have been proposed that guarantee termination on every database instance [21, 9, 22, 18]. The common idea is to statically assert that there are no positions in the database schema where fresh null values might be cyclically created in. The term *position* refers to a position in a relational predicate, e.g. $E(\textit{src}, \textit{dst})$ has two positions, namely *src*, denoted as E^1, and *dst*, denoted

*The work of this author was funded by Deutsche Forschungsgemeinschaft grant GRK 806/03.

VLDB '09, August 24-28, 2009, Lyon, France

as E^2. Likewise, we denote by S^1 the only position in predicate S. The non-terminating chase sequence discussed before, for instance, cyclically creates fresh null values in positions E^1 and S^1.

One well-known termination condition is *weak acyclicity* [21]. Roughly spoken, it implements a global study of the input constraints, to detect cyclically connected positions in the constraint set that introduce some fresh null values. In [9], the latter condition was generalized to a condition called *stratification*, showing that it suffices to assert weak acyclicity locally for subsets of constraints that might cyclically cause to fire each other. It is important to notice that the techniques introduced in [21, 9] take only the constraints into account and not the database instance. We therefore call such termination conditions *data-independent*; their result is either the guarantee that the chase with these constraints terminates for *every* database instance or that no predictions can be made.

This paper explores sufficient termination conditions beyond stratification, which (by the best of our knowledge) is the most general termination condition known so far. As one major contribution, we study data-independent chase termination and present conditions that generalize stratification. Complementary, we consider the novel problem of data-dependent chase termination, where our goal is to derive chase termination guarantees w.r.t. a fixed instance. In the remainder of the Introduction we summarize the key concepts and ideas of our analysis and survey the main results.

Data-independent chase termination. As discussed before, the source of non-terminating chase sequences are fresh null values that are cyclically created at runtime in some position(s). We develop new techniques that allow us to statically approximate the set of positions where null values are created in or copied to during chase application and use them to develop a hierarchy of sufficient termination conditions that are strictly more general than stratification. Our termination conditions rely on the following ideas.

(1) Identification of harmless null values: Often constraints introduce fresh null values in a certain position, but the (fixed) size of the database instance implies an upper bound on the number of null values that might be introduced in this position. Consider for example the constraint $\alpha_3 := \forall x, y(S(x), E(x, y) \rightarrow \exists z E(z, x))$, which may create fresh null values in position E^1. Whenever α_3 is part of a constraint set that does not copy null values to or create null values in position S^1, the number of fresh null values that might be introduced in position E^1 by α_3 is implicitly fixed by the number of entries in relation S and constraint α_3 cannot cause an infinite cascading of fresh null values in this position.

(2) Supervision of the flow of null values: We statically approximate the set of positions where null values might be copied to during chase application, by a sophisticated study of the interrelations between the individual constraints. Again, we illustrate the idea by a small and simple example. Let us consider the two constraints $\beta_1 := \forall x, y(S(x), E(x, y) \rightarrow E(y, x))$ and $\beta_2 := \forall x, y(S(x), E(x, y) \rightarrow \exists z E(y, z), E(z, x))$, which assert that each special node with an outgoing edge has cycles of length 2 and 3, respectively. We observe that none of these constraints inserts fresh null values into relation S, so the chase will terminate as soon as β_1 and β_2 have been fixed for all special nodes with an outgoing edge, i.e. after a finite number of steps. Somewhat surprisingly, none of the existing conditions recognizes chase termination for the above scenario. The reason is that they do not supervise the flow of null values. Our approach exhibits such an analysis and would guarantee chase termination for the two constraints above.

(3) Inductive decomposition of the constraint set: The constraint set in the previous example is not dangerous, because no fresh null values are created in position S^1. Let us, in addition to β_1 and β_2, consider the constraint $\beta_3 := \exists x, y S(x), E(x, y)$, stating that there is at least one special node with an outgoing edge. Clearly, β_3 fires at most once, so the chase for the constraint set $\{\beta_1, \beta_2, \beta_3\}$ will still terminate. However, β_3 complicates the analysis because it "infects" position S^1 in the sense that now null values may be created in this position. We resolve such situations by an (inductive) decomposition of the constraint set. When applied to the above example, our approach would recognize that β_3 is not cyclically connected with β_1 and β_2, and decompose the constraint set into the subsets $\{\beta_1, \beta_2\}$ and $\{\beta_3\}$, which then are inspected recursively.

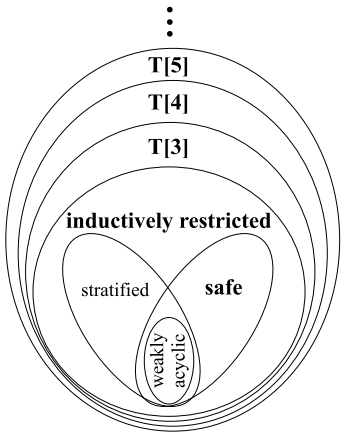

Figure 1: Chase termination conditions.

Based upon the previous ideas we develop two novel sufficient chase termination condition, called *safety* and *inductive restriction*. Figure 1 surveys our main results and relates them to the previous termination conditions weak acyclicity and stratification. All classes in the figure guarantee chase termination in polynomial-time data complexity and all inclusion relationships are strict. As can be seen, safety generalizes weak acyclicity and is further generalized by inductive restriction. On top of inductively restricted constraints we then define an (infinite) hierarchy of sufficient termination conditions, which we call T-hierarchy. To give an intuition, for a fixed level in this hierarchy, say $T[k]$, the idea is to study the flow and creation of fresh null values detailedly for chains of up to k constraints that might cause to fire each other in sequence.

An algorithm. It can be checked in polynomial time if a constraint set is safe; in contrast, the recognition problem for inductively restricted constraints and the classes in the T-hierarchy is in coNP. We develop an efficient algorithm that accounts for the increasing complexity of the recognition problem and can be used to test membership of a constraint set in some fixed level of the T-hierarchy. The underlying idea idea of our algorithm is to combine the different sufficient termination conditions, to reduce the complexity of checking for termination wherever possible.

Data-dependent chase termination. Whenever the input constraint set does not fall into some fixed level of the T-hierarchy, no termination guarantees for the general case can be derived. Arguably, reasonable applications should never risk non-termination, so the chase cannot be safely applied to *any instance* in this case. Tackling this situation, we study the novel problem of *data-dependent chase termination:* given constraint set Σ and a fixed instance I, does the chase with Σ terminate on I? We argue that this setting particularly makes sense in the context of Semantic Query Optimization, where the query – interpreted as database instance – is chased: typically, the query is small, so the "data" part can be analyzed efficiently (as opposed to the case where the input is a large

database instance). We propose two complementary approaches:

1. Our first, static scheme relies on the observation that, if the instance is fixed, we can ignore constraints in the constraint set that will never fire when chasing the instance, i.e. if general sufficient termination guarantees hold for those constraints that might fire. As a fundamental result, we show that in general it is undecidable if a constraint will never fire on a fixed instance. Still, we give a *sufficient* condition that allows us to identify such constraints in many cases and derive a sufficient data-dependent condition.

2. Whenever the static approach fails, our second, dynamic approach comes into play: we run the chase and track cyclically created fresh null values in a so-called monitor graph. We then fix the maximum depth of cycles in the monitor graph and stop the chase when this limit is exceeded: in such a case, no termination guarantees can be made. However, we show that each fixed search depth implicitly defines a class of constraint-instance pairs for which the chase terminates. Intuitively, the search depth limit can be seen as a natural condition that allows us to stop the chase when "dangerous" situations arise. Under these considerations, our approach adheres to situations that are likely to cause non-termination, so it is preferable to blindly running the chase and aborting after a fixed amount of time, or a fixed number of chase steps. Applications might fix the search depth following a pay-as-you-go principle. Ultimately, the combination of our static and dynamic analysis constitutes a pragmatic workaround in all scenarios where no general (i.e., data-independent) termination guarantees can be made.

Application. As a possible application of our techniques, we review the problem of answering Conjunctive Queries over knowledge bases in the presence of constraints, with a focus on scenarios where the chase with the given constraint set does not necessarily terminate. This problem was first considered in [13] and recently generalized in [5, 6]. A key idea in [5] is an overestimation of the set of positions in which null values might occur, using the concept of so-called affected positions. In particular, affected positions are used in [5] to define a class of constraints called weakly guarded constraint sets, for which the query answering problem is decidable. Using our novel techniques, we refine the notion of affected positions with the help of a so-called restriction system, which is a central tool in our study of data-independent chase termination, e.g. used to define the class of inductively restricted constraints and the T-hierarchy. We show that restriction systems can be fruitfully applied to generalize the class of weakly guarded constraints to a class we call restrictedly guarded constraints, thus making the algorithms in [5, 6] applicable to a larger class of constraints.

Structure. Section 2 presents the necessary background in databases. Next, Section 3 provides our results on data-independent chase termination. Its main results are the introduction of the T-hierarchy and an algorithm to efficiently test membership of a constraint set in some level of the T-hierarchy. In Section 4 we then motivate the novel problem of data-dependent chase termination. As a possible application, Section 5 demonstrates the applicability of our concepts and methods in the context of query answering on knowledge bases where the chase may not terminate. We conclude with some closing remarks in Section 6.

Additional remarks. This paper builds upon the ideas presented in the Extended Abstract [17]. Other parts of this paper were informally published as technical reports [22, 18].

2. PRELIMINARIES

General mathematical notation. The natural numbers $\mathbb{N}$ do not include 0. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, ..., n\}$. For a set M, we denote by 2^M its powerset and by $|M|$ its cardinality. Abusing notation we denote by $|s|$ also the length of a logical formula. Given a tuple $t = (t_1, \ldots, t_n)$ we define the tuple obtained by projecting on positions $1 \leq i_1 < \cdots < i_m \leq n$ as $p_{i_1,\ldots,i_m}(t) := (t_{i_1}, \ldots, t_{i_m})$.

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. Often we will denote a sequence of variables, constants or labeled nulls by $\overline{a}$ if the length of this sequence is understood from the context. A *database schema* $\mathcal{R}$ is a finite set of relational symbols $\{R_1, ..., R_n\}$. To every relational symbol $R \in \mathcal{R}$ we assign a natural number $ar(R)$ called its *arity*. A database position is a pair (R, i) where $R \in \mathcal{R}$ and $i \in [ar(R)]$, for short we write R^i, e.g. a three-ary predicate S has three positions S^1, S^2, S^3. We say that a variable, labeled null, or constant c appears e.g. in position R^1 if there exists an atom $R(c, ...)$. In the rest of the paper, we assume the database schema and the set of constants and labeled nulls to be fixed and therefore we will suppress them in our notations. A *database instance* I is a finite set of $\mathcal{R}$-atoms that contains only elements from $\Delta \cup \Delta_{null}$ in its positions. The domain of I, $dom(I)$, is the set of elements from $\Delta \cup \Delta_{null}$ that appear in I.

Conjunctive Queries. A Conjunctive Query (CQ) is an expression of the form $ans(\overline{x}) \leftarrow \varphi(\overline{x}, \overline{z})$, where φ is a conjunction of relational atoms, $\overline{x}$, $\overline{z}$ are sequences of variables and constants, and it holds that every variable in $\overline{x}$ also occurs in φ. If $\overline{x}$ is empty we call the query boolean. The semantics of such a query q on database instance I is defined as $q(I) := \{\ \overline{a} \in \Delta^{|\overline{x}|} \mid I \models \exists \overline{z} \varphi(\overline{a}, \overline{z})\ \}$.

Constraints. Let $\overline{x}, \overline{y}$ be sequences of variables. We consider two types of database constraints: *tuple generating dependencies* (TGDs) and *equality generating dependencies* (EGDs). A TGD is a first-order sentence $\alpha := \forall \overline{x}(\phi(\overline{x}) \rightarrow \exists \overline{y} \psi(\overline{x}, \overline{y}))$ such that (a) both ϕ and ψ are conjunctions of atomic formulas (possibly with parameters from Δ), (b) ψ is not empty, (c) ϕ is possibly empty, (d) both ϕ and ψ do not contain equality atoms and (e) 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 is a first-order sentence $\alpha := \forall \overline{x}(\phi(\overline{x}) \rightarrow x_i = x_j)$, 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 the set of positions in ϕ by $pos(\alpha)$.

From now on we will use the word constraint instead of saying that a logical expression may be a TGD or an EGD. Satisfaction of constraints by databases is defined in the standard first-order manner and is therefore omitted here. We write $I \models \alpha$ if a constraint α is satisfied by I and $I \not\models \alpha$ otherwise. 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 $pos(\Sigma) := \bigcup_{\xi \in \Sigma} pos(\xi)$. We use the term $body(\alpha)$ for a constraint α as the set of atoms in its premise; analogously we define $head(\alpha)$. In case α is a constraint and $\overline{a}$ is a sequence of labeled nulls and constants, then $\alpha(\overline{a})$ is the constraint α without universal quantifiers but with parameters $\overline{a}$. We will often abuse this notation and say that a labeled null occurs in $\alpha(\overline{a})$, meaning that a labeled null is the parameter for some universally quantified variable in α.

Homomorphisms. A homomorphism from a set of atoms A_1 to a set of atoms A_2 is a mapping $\mu : \Delta \cup V \rightarrow \Delta \cup \Delta_{null}$ such that the following conditions hold: (i) if $c \in \Delta$, then $\mu(c) = c$ and (ii) if $R(c_1, ..., c_n) \in A_1$, then $R(\mu(c_1), ..., \mu(c_n)) \in A_2$.

Chase. Let Σ be a set of TGDs and EGDs and I an instance, represented as a set of atoms. We say that a TGD $\forall \overline{x} \varphi \in \Sigma$ is applicable to I if there is a homomorphism μ from $body(\forall \overline{x} \varphi)$

Schema: $S(n)$, $E(src, dst)$
Constraint Set: $\Sigma := \{\alpha\}$, where

α : If x_2 is a special node and has some predecessor x_1, then x_1 has itself a predecessor: $S(x_2), E(x_1, x_2) \rightarrow \exists y\, E(y, x_1)$

Figure 2: A sample constraint.

to I and μ cannot be extended to a homomorphism $\mu' \supseteq \mu$ from $head(\forall\overline{x}\varphi)$ to I. In such a case the chase step $I \xrightarrow{\forall\overline{x}\varphi,\mu(\overline{x})} J$ is defined as follows. We define a homomorphism ν as follows: (a) ν agrees with μ on all universally quantified variables in φ, (b) for every existentially quantified variable y in $\forall\overline{x}\varphi$ we choose a "fresh" labeled null $n_y \in \Delta_{null}$ and define $\nu(y) := n_y$. We set J to be $I \cup \nu(head(\forall\overline{x}\varphi))$. We say that an EGD $\forall\overline{x}\varphi \in \Sigma$ is applicable to I if there is a homomorphism μ from $body(\forall\overline{x}\varphi)$ to I and it holds that $\mu(x_i) \neq \mu(x_j)$. In such a case the chase step $I \xrightarrow{\forall\overline{x}\varphi,\mu(\overline{x})} J$ is defined as follows. We set J to be

- I except that all occurrences of $\mu(x_j)$ are substituted by $\mu(x_i) =: a$, if $\mu(x_j)$ is a labeled null,
- I except that all occurrences of $\mu(x_i)$ are substituted by $\mu(x_j) =: a$, if $\mu(x_i)$ is a labeled null,
- undefined, if both $\mu(x_j)$ and $\mu(x_i)$ are constants. In this case we say that the chase fails.

A chase sequence is an exhaustive application of applicable constraints $I_0 \xrightarrow{\varphi_0,\overline{a}_0} I_1 \xrightarrow{\varphi_1,\overline{a}_1} \ldots$, where we impose no strict order what constraint must be applied in case several constraints apply. 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 of applicable constraints may lead to a different chase result. However, as proven in [21], two different chase orders 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 [8, 4, 13] that $I^\Sigma \models \Sigma$. In case that a chase step cannot be performed (e.g., because a homomorphism would have to equate two constants) the chase result is undefined. If we have an infinite chase sequence $I_0 \xrightarrow{\varphi_0,\overline{a}_0} I_1 \xrightarrow{\varphi_1,\overline{a}_1} \ldots$, we distinguish two cases: (i) if the constraint set contains an EGD, then we also say that the result is undefined; (ii) if the constraint set consists of TGDs only then $I^\Sigma := \bigcup_{i\geq 0} I_i$ is the union of all intermediate database instances during the application of the chase.

3. DATA-INDEPENDENT TERMINATION

In this section we discuss the sufficient data-independent chase termination conditions presented in Figure 1. First, we will review existing approaches and then introduce the novel class of *safe* constraints, which strictly generalizes weak acyclicity, but is different from stratification. Building upon the definition of safety, we then introduce *inductively restricted* constraints as a consequent advancement of our ideas. The latter class strictly subsumes all termination conditions known so far. Finally, we will define a hierarchy of sufficient termination condition on top of inductively restricted constraints, the so-called T-hierarchy. Each level $T[k]$ in this hierarchy is strictly contained in the next level $T[k+1]$. Our novel sufficient termination conditions vastly extend the applicability of the chase algorithm, as they guarantee chase termination for much larger classes of constraints than previous conditions.

As a minimalistic motivating example for our study of novel chase termination conditions let us consider the constraint set Σ from Figure 2, which is settled in our graph database schema from the Introduction. As we shall see later, the chase with Σ terminates for every database instance. Still, none of the existing termination conditions is able to recognize termination for this constraint set, i.e. Σ is neither weakly acyclic nor stratified. With the techniques and tools that we develop within this section, we will be able to guarantee chase termination for Σ on every database instance.

3.1 Weak Acyclicity

The notion of weak acyclicity from [10, 21] is the starting point for our discussion. Informally spoken, the key idea of weak acyclicity is to statically estimate the flow of data between the database positions during the execution of the chase. Weak acyclicity asserts that no fresh values are created over and over again.

DEFINITION 1. (see [21]) The dependency graph dep(Σ) of a set of constraints Σ is the directed graph defined as follows. The set of vertices is the set of positions that occur in some TGD in Σ. There are two kinds of edges. 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

- 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).
- for every existentially quantified variable y and for every occurrence of y in a position π_2, add a *special* edge $\pi_1 \xrightarrow{*} \pi_2$ (if it does not already exist).

A set Σ of TGDs and EGDs is called *weakly acyclic* iff dep(Σ) has no cycles going through a special edge. □

Intuitively, normal edges in the dependency graph track the flow of data between the database positions and special edges cover the case of newly introduced null values. If the dependency graph contains no cycles through a special edge it cannot happen that fresh null values are cyclically added to the database instance. It has been shown in [21] that weak acyclicity can be decided in polynomial time. We illustrate the definition of weak acyclicity by example.

EXAMPLE 1. We depict the dependency graph for the constraint set $\Sigma := \{\alpha_1, \alpha_2, \alpha_3\}$ from Figure 7 in Figure 3. One can observe that Σ is not weakly acyclic, as witnessed by the self-loop through special edge $fly^2 \xrightarrow{*} fly^2$. □

3.2 Stratification

In [9], stratification was set on top of weak acyclicity. The main idea behind stratification is to decompose the constraint set into independent subsets that are then separately tested for weak acyclicity. More precisely, the decomposition splits the input constraint set into subsets of constraints that may cyclically cause to fire each other. The termination guarantee for the full constraint set follows if weak acyclicity holds for each subset in the decomposition.

DEFINITION 2. (see [9]) Given two TGDs or EGDs $\alpha, \beta \in \Sigma$ we define $\alpha \prec \beta$ iff there exists a relational database instance I and $\overline{a}, \overline{b}$ such that (i) $I \not\models \alpha(\overline{a})$, (ii) $I \models \beta(\overline{b})$, (iii) $I \xrightarrow{\alpha,\overline{a}} J$, and (iv) $J \not\models \beta(\overline{b})$. □

Intuitively, $\alpha \prec \beta$ means that if α fires it can cause β to fire (in the case that β could not fire before).

EXAMPLE 2. (see [9]) Let predicate E store the edge relation of a graph and let the constraint $\alpha := E(x_1, x_2), E(x_2, x_1) \rightarrow$

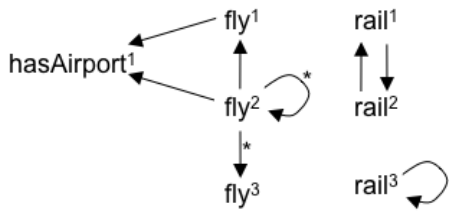

Figure 3: Dependency graph for Σ from Figure 7.

$\exists y_1, y_2 E(x_1, y_1), E(y_1, y_2), E(y_2, x_1)$ be given, stating that each node having a cycle of length 2 also has a cycle of length 3. A 3-cycle can never be a 2-cycle again, so it holds that $\alpha \not\prec \alpha$. □

DEFINITION 3. (see [9]) The chase graph $G(\Sigma) = (\Sigma, E)$ of a set of constraints Σ contains a directed edge (α, β) between two constraints iff $\alpha \prec \beta$. We call Σ stratified iff the constraints in every cycle of $G(\Sigma)$ are weakly acyclic. □

EXAMPLE 3. Consider the constraint α from Example 2. It holds that $\alpha \not\prec \alpha$, so $\{\alpha\}$ is stratified. As shown in [9], the dependency graph of $\{\alpha\}$ contains a cycle through a special edge, so $\{\alpha\}$ is not weakly acyclic. □

It can be decided in coNP whether a set of constraints is stratified. Like weak acyclicity, stratification guarantees the termination of the chase in polynomial time data complexity (see [9]), i.e. the set of constraints is fixed and the number of chase steps is polynomial in the number of distinct values in the input database instance. Stratification strictly generalizes weak acyclicity, thus (i) if Σ is weakly acyclic, then it is also stratified and (ii) there are constraint sets that are stratified but not weakly acyclic (cf. Example 3).

3.3 Safety

The basic idea of our first new termination condition, *safety*, is to estimate the set of positions where labeled nulls may be copied to and (statically) analyze the data flow only between those positions. As a useful tool, we borrow the notion of so-called *affected positions* from [5], which is an overestimation of the positions in which a null value that was introduced during the chase may occur.

DEFINITION 4. *[5]* Let Σ be a set of TGDs. The set of affected positions aff(Σ) of Σ 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 $\pi \in$ aff(Σ).
- If the same universally quantified variable X appears both in position π and only in affected positions in the body of α, then $\pi \in$ aff(Σ). □

Although we borrow this definition from [5], our focus is different. We use affected positions to extend known classes of constraints for which the chase terminates, whereas [5] investigates query answering in cases the chase may not terminate. Our work neither subsumes [5] nor the other way around.

We motivate the safety termination condition using the single constraint $\beta := R(x_1, x_2, x_3), S(x_2) \rightarrow \exists y R(x_2, y, x_1)$. The dependency graph of constraint set $\{\beta\}$ is shown in Figure 4 (left). As can be seen, there is a cycle going through a special edge, so the set is not weakly acyclic. We next study the affected positions in β:

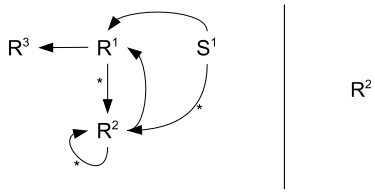

Figure 4: Left: Dependency graph. Right: Corresponding propagation graph (it has no edges).

EXAMPLE 4. Let us consider the constraint set $\Sigma := \{\beta\}$. Clearly, position R^2 is affected because it contains an existentially quantified variable. S^1 is not affected because S is not modified when chasing with the single constraint β. Finally, we observe that also R^1 is not affected because x_2 occurs not only in R^2 but also in S^1, which is not an affected position. We conclude that position R^2 is the only affected position in constraint set Σ. □

We now argue that for constraint β a cascading of fresh labeled nulls cannot occur, i.e. no fresh labeled null can repeatedly create new labeled nulls in position R^2 while copying itself to position R^1. The reason is that β cannot be violated with a fresh labeled null in R^2, i.e. if $R(a_1, a_2, a_3)$ and $S(a_2)$ hold, but $\exists y R(a_2, y, a_1)$ does not, then a_2 is never a newly created labeled null. This is due to the fact that a_2 also occurs in S^1, but S^1 is not an affected position. Hence, the chase sequence always terminates. We will later see that this is not a mere coincidence: the constraint is safe.

Like in the case of weak acyclicity, we define the safety condition with the help of the absence of cycles containing special edges in some graph, called propagation graph.

DEFINITION 5. Given a set of TGDs Σ, the propagation graph prop(Σ) := (aff(Σ), E) is the directed graph defined 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). □

As an improvement over weak acyclicity, in the propagation graph we do not supervise the whole data flow but only the flow of labeled nulls that might be introduced at runtime. Consequently, the graph contains edges only for null values that stem exclusively from affected positions. We now can easily define the safety condition on top of the propagation graph.

DEFINITION 6. A set Σ of constraints is called *safe* iff prop(Σ) has no cycles going through a special edge. □

EXAMPLE 5. Consider the constraint β from Example 4. Its dependency graph is depicted in Figure 4 on the left side and its propagation graph on the right side. The latter contains only the affected position R^2 (and no edges). From Definitions 1 and 6 it follows that β is safe, but not weakly acyclic. □

The intuition of safety is that we forbid an unrestricted cascading of null values, i.e. with the help of the propagation graph we impose

a partial order on the affected positions such that any newly introduced null value can only be created in a position that has a higher rank in that partial order in comparison to null values that may occur in the body of a TGD. To state this more precisely, assume that a TGD of the form $\forall \overline{x}(\phi(\overline{x}) \rightarrow \exists \overline{y}\psi(\overline{x}, \overline{y}))$ is violated. Then, $I \models \phi(\overline{a})$ and $I \not\models \exists \overline{y}\psi(\overline{a}, \overline{y}))$ must hold. The safety condition ensures that any position in the body that contains a newly created labeled null from $\overline{a}$ and occurs in the head of the TGD has a strictly lower rank in our partial order than any position in which some element from $\overline{y}$ occurs. The main difference compared to weak acyclicity is that, in safety, we look in a refined way (cf. affected positions) on positions where labeled nulls can be propagated to.

It is easy to see that, if a constraint set Σ is safe, then every subset of Σ is safe, too. Furthermore, we note that, given a set of constraints, it can be decided in polynomial time if it is safe or not. In the following theorem we relate safety to the previous termination conditions weak acyclicity and stratification. In particular, the theorem clarifies the observation from Example 5, where we could observe that the propagation graph is a subgraph of the dependency graph. This is not a mere coincidence:

THEOREM 1. Let Σ be a set of constraints.

- The graph prop(Σ) is a subgraph of dep(Σ).
- If Σ is weakly acyclic, then it is also safe.
- There is some Σ that is safe, but not stratified and vice versa. □

Proof Sketch. (a) The set of vertices from prop(Σ) is contained in the set of vertices of dep(Σ). In order to add an edge to prop(Σ) stronger prerequisites must be fulfilled than in the construction of dep(Σ). Therefore prop(Σ) is a subgraph of dep(Σ). (b) If dep(Σ) does not have a cycle through a special edge, then prop(Σ) cannot have. (c) Let $\alpha := S(x_2, x_3), R(x_1, x_2, x_3) \rightarrow \exists y R(x_2, y, x_1)$ and $\beta := R(x_1, x_2, x_3) \rightarrow S(x_1, x_3)$. It can be seen that $\alpha \prec \beta$ and $\beta \prec \alpha$. Together with the fact that $\{\alpha, \beta\}$ is not weakly acyclic it follows that $\{\alpha, \beta\}$ is not stratified. However, $\{\alpha, \beta\}$ is safe. Let $\gamma := T(x_1, x_2), T(x_2, x_1) \rightarrow \exists\ y_1, y_2\ T(x_1, y_1), T(y_1, y_2), T(y_2, x_1)$ (see [9]). It was argued in [9] that $\{\gamma\}$ is stratified. However, it is not safe because both T^1 and T^2 are affected. Therefore we have that dep($\{\gamma\}$) = prop($\{\gamma\}$) and it was argued in [9] that it is not weakly acyclic. □

Like stratification and weak acyclicity, safety guarantees the termination of the chase in polynomial time data complexity, i.e. the set of constraints is fixed and the number of chase steps is polynomial in the number of distinct values in the input database instance:

THEOREM 2. Let Σ be a fixed set of safe 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 $Q(|dom(I)|)$. □

We omit the proof of the theorem for space limitations, referring the interested reader to the technical report [16].

3.4 Inductive Restriction

In this section we generalize the method that lifts weak acyclicity to stratification from [9] with the help of so-called *restriction systems*. The chase graph from [9] will be a special case of such a restriction system. With the help of restriction systems we then define a new sufficient termination condition called *inductive restriction*, whose main idea is to decompose a given constraint set into smaller subsets (in a more refined way than stratification). We then use the safety condition from before to check the termination of every subset and, whenever all subsets are safe, the termination for the full constraint set can be guaranteed. Ultimately, we show that inductive restriction (like all the classes discussed before) guarantees chase termination in polynomial-time data complexity. This section also lays the foundations for the T-hierarchy (cf. Figure 1), which will be defined subsequently in Section 3.5. We motivate our study with a constraint set that is neither safe nor stratified.

EXAMPLE 6. Let predicate E(x,y) store graph edges and predicate S(x) store some nodes. The constraints set $\Sigma = \{\alpha_1, \alpha_2\}$ 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 cycles of length 2 and 3, respectively. It holds that aff(Σ) = $\{E^1, E^2\}$ and it is easy to verify that Σ is neither safe nor stratified. In particular, it we observe that $\alpha_1 \prec \alpha_2$ and $\alpha_2 \prec \alpha_1$. □

The first task in our formalization is a refinement of relation $\prec$ from [9]. This refinement will helps us to detect if during the chase null values might be copied to the head of some constraint. To simplify the definition, we introduce the notion of null-pos:

DEFINITION 7. Let Σ be a set of constraints, I be a fixed database instance and $A \subseteq \Delta_{null}$. Then, we define null-pos(A, I) as $\{\pi \in pos(\Sigma) \mid a \in A, a \text{ occurs in position } \pi \text{ in } I\}$. □

Informally spoken, null-pos(A, I) is the set of positions in I in which the elements (i.e., labeled nulls) from A occur. We are now ready to define the refinement of relation $\prec$:

DEFINITION 8. Let Σ be a set of constraints 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_0 such that

- $I_0 \overset{\alpha,\overline{a}}{\rightarrow} I_1 \overset{\beta,\overline{b}}{\rightarrow} I_2$,
- there is $n \in \overline{b} \cap \Delta_{null}$ in the head of $\beta(\overline{b})$ such that null-pos$(\{n\}, I_0) \subseteq P$, and
- $I_0 \models \beta(\overline{b})$. □

The refinement of $\prec$ forms the basis for the notion of a so-called restriction system, which is a strict generalization of the chase graph introduced in [9] and will serve as a central tool in our work. The two definitions below formalize restriction systems.

DEFINITION 9. 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. □

DEFINITION 10. A 2-*restriction system*[1] 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$(\alpha, f(\alpha)) \subseteq f(\beta)$,
- forall EGDs α and forall $(\alpha, \beta) \in E$: $f(\alpha) \subseteq f(\beta)$, and
- forall $\alpha, \beta \in \Sigma$: $\alpha \prec_{f(\alpha)} \beta \implies (\alpha, \beta) \in E$.

A 2-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. □

We illustrate this definition by two examples. The first one also shows that restriction systems always exist.

[1] In [18, 17, 22] the notion of a 2-restriction system was simply called restriction system and was defined slightly different there.

```
part(Σ: Set of TDGs and EGDs, k: not equal to 1) {
 1: compute the strongly connected components (as
    sets of constraints) C_1, ..., C_n of the minimal
    k-restriction system of Σ;
 2: D ← ∅
 3: if (n == 1) then
 4:       if (C_1 ≠ Σ) then
 5:             return part(C_1,k);
 6:       endif
 7:       return {Σ};
 8: endif
 6: for i=1 to n do
 9:       D ← D ∪ part(C_i,k);
10: endfor
11: return D; }
```

Figure 5: Algorithm to compute subsets of Σ.

EXAMPLE 7. Let Σ a set of constraints. Then, $(G(\Sigma), f)$, where $f(\alpha) := pos(\{\alpha\})$ for all $\alpha \in \Sigma$ is a 2-restriction system for constraint set Σ. □

EXAMPLE 8. Consider Σ from Example 6. The minimal 2-restriction system for Σ is G'(Σ):=(Σ,{(α_2,α_1)}) with f(α_1) := $\{E^1,E^2\}$ 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. □

Restriction systems are useful tools to define new classes of constraints that guarantee chase termination. To give an example, one can show that the chase with a constraint set Σ terminates for every database instance if every strongly connected component of its minimal 2-restriction system is safe. We refer the interested reader to [18] for details, where this class was formally introduced under the name *safe restriction*. Note that the constraint set Σ from Example 6 falls into the class of safely restricted constraints, because its minimal 2-restriction system (given in Example 8) contains no strongly connected component. In this work, we skip the formal definition of safe restriction, but instead go one step further and define a termination condition called *inductive restriction*, which further generalizes safe restriction. The following example provides a constraint set that is not safely restricted but, as we shall see later, falls into the class of inductively restricted constraints.

EXAMPLE 9. We extend the constraint set from Example 8 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(α_3) := $\emptyset$ is the minimal 2-restriction system. It contains the strongly connected component $\{\alpha_1,\alpha_2\}$. Note that Σ' is neither safe, nor stratified, nor safely restricted. Hence, using the sufficient termination conditions discussed so far no chase termination guarantees can be made for Σ'. □

Intuitively, in the example above the constraint α_3 "infects" position S^1 in the 2-restriction system. Still, 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 resolves such situations by recursively computing the minimal 2-restriction systems of the strongly connected components. We formalize this computation in Algorithm 1, called $part(\Sigma, 2)$ and define the class of inductively restricted constraint sets by help of this algorithm.

DEFINITION 11. Let Σ be a set of constraints. We call Σ *inductively restricted* iff every $\Sigma' \in part(\Sigma, 2)$ is safe. □

Compared to stratification, inductive restriction does not increase the complexity of the recognition problem:

LEMMA 1. Let Σ be a set of constraints. The recognition problem for inductive restriction is in coNP. □

Proof Sketch. We start with an additional claim: let P be a set of positions and α, β constraints. Then, the mapping $(P, \alpha, \beta) \mapsto \alpha \prec_P \beta$? can be computed by an NP-algorithm. The proof of this claim proceeds like the proof of Theorem 3 in [9]. It is enough to consider candidate databases for I_0 of size at most $|\alpha| + |\beta|$, i.e. unions of homomorphic images of the premises of α and β s.t. null values occur only in positions from P. Because of this claim, the minimal 2-restriction system of a set of constraints can be computed by an NP-algorithm (only polynomially many steps must be performed to reach the fixedpoint). Computing $part(\Sigma, 2)$ can also be done in non-deterministic polynomial time. To prove that Σ is not inductively restricted, guess some $\Sigma' \in part(\Sigma, 2)$ and verify that it is not safe. □

We give an example for an inductively restricted constraint set, which – as argued in Example 9 – is neither safe nor stratified.

EXAMPLE 10. Referring back to Example 9, we have seen that the minimal restriction system of Σ' contains the only strongly connected component $\{\alpha_1,\alpha_2\}$, which by Example 6 is not safe. Therefore we compute the minimal restriction system of $\{\alpha_1,\alpha_2\}$ and see that it does not contain a cycle. This argumentation proves that $part(\Sigma', 2) = \emptyset$, so we conclude that constraint set Σ' is inductively restricted. □

As depicted in Figure 1, the inductive restriction condition generalizes both safeness and stratification. The following proposition formally states these results and shows that the respective inclusion relationships are proper.

PROPOSITION 1. The following claims hold.

- If Σ is stratified, then it is also inductively restricted.
- If Σ is safe, then it is inductively restricted.
- There is some Σ that is inductively restricted, but neither safe nor stratified. □

Proof Sketch. We start with bullet one. It follows directly from the definition that if $\alpha \prec_P \beta$, then $\alpha \prec \beta$. Therefore it holds that every $\Sigma' \in part(\Sigma, 2)$ is contained in some strongly connected component of the chase graph of Σ. As every such strongly connected component is weakly acyclic, it is also safe. Consequently, also Σ' is safe. Bullet two follows from the fact that every subset of a safe constraint set is safe. Finally, bullet three is proven by the constraint set from Examples 9 and 10. □

The next theorem gives the main result concerning inductive restriction, showing that it guarantees chase termination in polynomial time data complexity. We refer the interested reader to our technical report [16] for a formal proof of this theorem.

THEOREM 3. 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 $Q(|dom(I)|)$. □

We conclude with the remark that our motivating constraint set from Figure 2 is not inductively restricted: the constraint α can

cause itself to fire, so its minimal 2-restriction system contains an edge from α to α, which forms a strongly connected component; further, α is not safe. To show that the chase with α terminates, we need weaker termination conditions than inductive restriction.

3.5 The T-Hierarchy

This section introduces the T-hierarchy, which is our main result regarding data-independent chase termination. Its lowest level, $T[2]$, corresponds to inductive restriction. Every level in the hierarchy is decidable and contains all lower levels. As we shall see, also the constraint from Figure 2 is a member of some level in this hierarchy. In the course of this section we leave out some proofs for space limitations, referring the interested reader to the technical report [16]. We start by defining the k-ary relation $\prec_{k,P}$ which is a generalization of $\prec_P$. The definition naturally extends the $\prec_P$ relation to a fixed number k of constraints.

DEFINITION 12. Let $k \geq 2$, Σ a set of constraints and $P \subseteq pos(\Sigma)$. For all $\alpha_1, ..., \alpha_k \in \Sigma$, we define $\prec_{k,P} (\alpha_1, ..., \alpha_k)$ iff there are tuples $\overline{a}_1, ..., \overline{a}_k$ and a database instance I_0 such that

- for all $i \in [k]$ it holds that $I_{i-1} \overset{\alpha_i, \overline{a}_i}{\rightarrow} I_i$,
- there is $n \in \overline{a}_k \cap \Delta_{null}$ in the head of $\alpha_k(\overline{a}_k)$ such that null-pos$(\{n\}, I_0) \subseteq P$,
- $I_0 \models \alpha_k(\overline{a}_k)$, and
- for every $i \in [k-1]$ there is $j \in [k] \backslash [i]$ such that $J_{j-1} \models \alpha_j(\overline{a}_j)$, where $J_0 := I_0$, $J_{l-1} \overset{\alpha_l, \overline{a}_l}{\rightarrow} J_l$ for $j > l \neq i$ and $J_i := J_{i-1}$. □

Note that $\prec_{2,P}$ corresponds exactly to $\prec_P$ introduced in Definition 8. It can be shown that, for a fixed value of k, membership in this relation is decidable in NP:

PROPOSITION 2. Let $k \geq 2$ be fixed. Then, there exists a NP-algorithm that decides for every set of positions P and every $\alpha_1, ..., \alpha_k \in \Sigma$ whether $\prec_{k,P} (\alpha_1, ..., \alpha_k)$ holds. □

The proof of this proposition proceeds like the proof that $\prec_P$ is decidable in NP time (cf. Lemma 1). We refer the interested reader to [16] for more details. We next use the relation $\prec_{k,P}$ to define k-restriction systems, which naturally generalize the 2-restriction systems defined over relation $\prec_P$ (cf. Definition 10).

DEFINITION 13. Let $k \in \mathbb{N}_{>1}$. A k-restriction system $G'_k(\Sigma)$ is a pair (G', f), where $G' = (\Sigma, E)$ is a graph and $f : \Sigma \rightarrow 2^{pos(\Sigma)}$ is a function such that

- forall TGDs α and forall $(\alpha, \beta) \in E$: aff-cl$(\alpha, f(\alpha)) \subseteq f(\beta)$,
- forall EGDs α and forall $(\alpha, \beta) \in E$: $f(\alpha) \subseteq f(\beta)$, and
- forall $\alpha_1, ..., \alpha_k \in \Sigma$: $\prec_{k,f(\alpha_1)} (\alpha_1, ..., \alpha_k)$ then $(\alpha_1, \alpha_2), ..., (\alpha_{k-1}, \alpha_k) \in E$.

A k-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) such that, 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. In case the third bullet is applied, E is extended. □

Note that for $k = 2$ this definition corresponds exactly to the definition of 2-restriction systems used to define inductive restriction. Like 2-restriction systems, minimal k-restriction systems are unique and can be computed by a coNP-algorithm:

PROPOSITION 3. Let $k \geq 2$ be fixed and Σ a set of constraints. The minimal k-restriction system for Σ is unique and can be computed by a NP-algorithm. □

We are now in the position to define the T-hierarchy:

DEFINITION 14. Let $k \geq 2$ and Σ be a set of constraints. Then $\Sigma \in T[k]$ iff there is $k' \in [k] \backslash \{1\}$ such that for every $\Sigma' \in part'(\Sigma, k')$ it holds that Σ' is safe. □

We call $T[k]$ the k-th level of the T-hierarchy. As a corollary from Proposition 3 we obtain that we can decide whether a set of constraints is in $T[k]$ by a coNP-algorithm. We next give an example for constraints in the T-hierarchy.

EXAMPLE 11. We set $\Sigma_{k+1} := \{\alpha_{k+1}\}$, where $\alpha_{k+1} := S(x_{k+1}), R_k(x_1, ..., x_{k+1}) \rightarrow \exists y R_k(y, x_1, ..., x_k)$. It holds that $\prec_{k,\emptyset} (\alpha, ..., \alpha)$ but not $\prec_{k+1,\emptyset} (\alpha, ..., \alpha)$. So the minimal $(k+1)$-restriction system does not contain any cycle, but the minimal k-restriction system does. Therefore $\Sigma_{k+1} \in T[k+1]$. On the other hand, we observe that the constraint is not safe, so it is not contained in $T[k]$. Also note that the constraint in Figure 2 exactly corresponds to Σ_2, so it is contained in level $T[3]$. □

The following proposition relates the levels of the T-hierarchy to each other and inductive restriction.

PROPOSITION 4. Let $k \geq 2$.

- Σ is inductively restricted iff $\Sigma \in T[2]$
- $T[k] \subseteq T[k+1]$.
- There is some Σ such that $\Sigma \in T[k+1] \backslash T[k]$. □

Proof Sketch. (a) To prove bullet one, note that both definitions coincide exactly. (b) Bullet two follows by definition. (c) For bullet three we refer back to Example 11. □

The next result is our main contribution concerning data-independent chase termination. It states that, for a fixed value of k, membership in $T[k]$ guarantees polynomial time data complexity for the chase. Again, the technical proof can be found in [16].

THEOREM 4. Let $k \geq 2$ and $\Sigma \in T[k]$ be a fixed set of 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 $Q(|dom(I)|)$. □

3.6 An Algorithmic Approach

This section aims to develop an efficient algorithm to test membership in $T[k]$. We have seen before that the computation of k-restriction systems is costly because we need NP time to compute the relation $\prec_{k,P}$. For this reason, we present an algorithm that avoids the computation of k-restriction systems where possible. It relies on the idea that (the weaker condition) safety can be checked in polynomial time (cf. Section 3.3). Before computing the k-restriction system, we always check for safety and, whenever safety holds, we conclude that the chase for the respective constraint set terminates and omit the k-restriction system computation.

To give a simple example, consider the constraint from Example 5, which has been shown to be safe, and assume we want to test if it falls into some (fixed) level k of the T-hierarchy. Computing a k-restriction system is superfluous, because membership in $T[k]$ trivially follows from the satisfaction of the safety condition.

In general, the situation is, of course, not that simple. Consider for instance the constraint set Σ' from Example 9 extended by $\{\alpha_4, \alpha_5\}$, where $\alpha_4 := E(x_1, x_2) \rightarrow T(x_1, x_2)$, $\alpha_5 := T(x_1, x_2) \rightarrow T(x_2, x_1)$, and call the resulting constraint set Σ''. Assume we want to show that Σ'' is inductively restricted (i.e., in $T[2]$). It follows from Example 6 that Σ'' is not safe. In direct correspondence to Example 9 it follows that the minimal 2-restriction system for

```
sub(Σ: Set of TDGs and EGDs, k: not equal to 1) {
 1: if (Σ is safe) then
 2:       return true;
 3: endif
 4: compute the strongly connected components (as
    sets of constraints) C_1, ..., C_n of the minimal
    k-restriction system of Σ;
 5: if (n == 0) then
 6:       return true;
 7: endif
 8: if (n == 1) then
 9:       if (C_1 ≠ Σ) then
10:             return check(C_1,k);
11:       endif
12:       return false;
13: endif
14: for i=1 to n do
15:       if (not check(C_i,k)) then
16:             return false;
17:       endif
18: endfor
19: return true; }

check(Σ: Set of TDGs and EGDs, k: not equal to 1) {
 1: for i = k downto 2 do
 2:       if (sub(Σ, i)) then return true;
 3: endfor
 4: return false; }
```

Figure 6: Algorithm to decide membership in $T[\cdot]$.

Σ'' is G'(Σ''):=(Σ'',{(α_1,α_2),(α_2,α_1),(α_3,α_1),(α_3,α_2),(α_1,α_4),(α_2,α_4),(α_4,α_5),(α_5,α_5)}), where f(α_1) = f(α_2) := {E^1,E^2,S^1}, f(α_3) := $\emptyset$, f(α_4) := {E^1,E^2} and f(α_5) := {T^1,T^2}. This 2-restriction system contains the strongly connected components {α_1,α_2} and {α_5}. For {α_1,α_2} we must compute its minimal 2-restriction system because it is not safe, but for {α_5} we can avoid this complexity because we know that α_5 is safe (indeed it is a full TGD) and therefore the chase terminates. We implement the scheme described above in algorithm *check*, provided in Figure 6.

PROPOSITION 5. Algorithm $check$ terminates and correctly decides membership in the T-hierarchy, i.e. $check(\Sigma, k)$ returns true if and only if $\Sigma \in T[k]$. □

Proof Sketch. The algorithm terminates because all recursive calls are made on constraint sets with size smaller than the input constraint set. What the algorithm does is trying to avoid the computation of k-restriction systems by testing for safeness. The correctness follows from the proof of Theorem 4 because the only property we need to show is that for all $\Sigma' \in part(\Sigma, k)$ the chase terminates, which is ensured by the additional safety checks. □

4. DATA-DEPENDENT TERMINATION

So far, we discussed conditions that guarantee chase termination for every database instance. In this section, we study the problem of data-dependent termination, i.e. given a constraint set Σ and a *fixed* instance I, does the chase with Σ terminate on I? By the best of our knowledge, this problem has not been studied before. Therefore, we start our discussion with a motivating scenario. Let us consider the travel agency database in Figure 7, where predicate *hasAirport* contains cities that have an airport and *fly* (*rail*) stores flight (rail) connections between cities, including their distance *dist*. In addition to the schema, constraints α_1-α_3 have been specified. For instance, α_3 might have been added to assert that, for each city reachable via plane, the schedule is integrated in the local database. Now consider the CQ q_1 below.

Sample Schema: *hasAirport*(*c_id*)
fly(*c_id1*, *c_id2*, *dist*)
rail(*c_id1*, *c_id2*, *dist*)

Constraint Set: $\Sigma = \{\alpha_1, \alpha_2, \alpha_3\}$, where

α_1 : If there is a flight connection between two cities, both of them have an airport:
$fly(c_1, c_2, d) \rightarrow hasAirport(c_1), hasAirport(c_2)$

α_2 : Rail-connections are symmetrical:
$rail(c_1, c_2, d) \rightarrow rail(c_2, c_1, d)$

α_3 : Each city that is reachable via plane has at least one outgoing flight scheduled:
$fly(c_1, c_2, d) \rightarrow \exists c_3, d' fly(c_2, c_3, d')$

Figure 7: Sample database schema and constraints.

$$q_1\colon\ rf(x_2) \leftarrow rail(c_1, x_1, y_1), fly(x_1, x_2, y_2)$$

The query selects all cities that can be reached from c_1 through rail-and-fly. Assume that, in the style of semantic query optimization, we want to optimize q_1 under constraints Σ using the chase. We then interpret the body of q_1 as database instance $I := \{rail(c_1, x_1, y_1), fly(x_1, x_2, y_2)\}$, where c_1 is a constant and the x_i, y_i labeled nulls. We observe that α_3 does not hold on I, since there is a flight to city x_2, but no outgoing flight from x_2. Hence, the chase adds a new tuple $t_1 := fly(x_2, x_3, y_3)$ to I, where x_3, y_3 are fresh labeled null values. In the resulting instance $I' := I \cup \{t_1\}$, α_3 is again violated (this time for x_3) and in subsequent steps the chase adds $fly(x_3, x_4, y_4)$, $fly(x_4, x_5, y_5)$, $fly(x_5, x_6, y_6)$, ... Obviously, it will never terminate.

Arguably, reasonable applications should never risk non-termination. It is clear, though, that the existence of (even a single) non-terminating chase sequences also means that no data-independent termination condition holds. Hence, based on data-independent conditions no query at all could be safely chased with the constraint set from Figure 7 and the benefit of the chase algorithm would be completely lost.[2] Despite the fact that there is a non-terminating chase sequence, however, there might be queries for which the chase with the constraint set from Figure 7 terminates. Tackling such situations, we propose to investigate data-dependent chase termination, i.e. to study sufficient termination guarantees for a *fixed instance* when no general termination guarantees apply. We illustrate the benefits of having such guarantees for query q_2 below, which selects all cities x_2 that can be reached from c_1 via rail-and-fly and the same transport route leads back from x_2 to c_1 (where c_1 is a constant and the x_i, y_i are variables).

$$q_2\colon\ rffr(x_2) \leftarrow rail(c_1, x_1, y_1), fly(x_1, x_2, y_2), fly(x_2, x_1, y_2), rail(x_1, c_1, y_1)$$

Query q_2 violates only α_1. It is easy to verify that the chase terminates for this query and transforms q_2 into q_2':

[2]Note that, principally, query optimization could also be done with a bounded portion of the chase result, but in general we do not find minimal rewritings of the input query in the style of [1]. Therefore, it is desirable to guarantee chase termination.

q_2': $\mathit{rffr}(x_2) \leftarrow \mathit{rail}(c_1, x_1, y_1), \mathit{fly}(x_1, x_2, y_2),$
$\mathit{fly}(x_2, x_1, y_2), \mathit{rail}(x_1, c_1, y_1),$
$\mathit{hasAirport}(x_1), \mathit{hasAirport}(x_2)$

The resulting query q_2' satisfies all constraints and is a so-called *universal plan* [1]: intuitively, it incorporates all possible ways to answer the query. As discussed in [1], the universal plan forms the basis for finding smaller equivalent queries (under the respective constraints), by choosing any subquery of q_2' and testing if it can be chased to a homomorphical copy of q_2'. Using this technique we can easily show that the following two queries are equivalent to q_2.

q_2'': $\mathit{rffr}(x_2) \leftarrow \mathit{rail}(c_1, x_1, y_1), \mathit{fly}(x_1, x_2, y_2),$
$\mathit{fly}(x_2, x_1, y_2)$
q_2''': $\mathit{rffr}(x_2) \leftarrow \mathit{hasAirport}(x_1), \mathit{rail}(c_1, x_1, y_1),$
$\mathit{fly}(x_1, x_2, y_2), \mathit{fly}(x_2, x_1, y_2)$

Instead of q_2 we thus could evaluate q_2'' or q_2''', which might well be more performant: in both q_2'' and q_2''' the join with $\mathit{rail}(x_1, c_1, y_1)$ has been eliminated; moreover, if *hasAirport* is duplicate-free, the additional join of *rail* with *hasAirport* in q_2''' may serve as a filter that decreases the size of intermediate results and speeds up query evaluation. This strategy is called *join introduction* in SQO (cf. [14]). Ultimately, the chase for q_2 made it possible to detect q_2'' and q_2''', so it would be desirable to have data-dependent termination guarantees that allow us to chase q_2 (and q_2'', q_2'''). We will present such conditions in the remainder of this section.

4.1 Static Termination Guarantees

Our first approach to data-dependent chase termination is a static one. It relies on the observation that the chase will always terminate on instance I if the subset of constraints that might fire when chasing I with Σ is contained in some level of the T-hierarchy. We call a constraint $\alpha \in \Sigma$ (I, Σ)-*irrelevant* if and only if there is no chase sequence such that α can eventually fire, i.e. no chase sequence of the form $I \xrightarrow{\alpha_1, \overline{a_1}} \cdots \xrightarrow{\alpha, \overline{a}} \ldots$.

LEMMA 2. Let $k \geq 2$ and $\Sigma' \subseteq \Sigma$ s.t. $\Sigma \setminus \Sigma'$ is a set of (I, Σ)-irrelevant constraints. If $\Sigma' \in T[k]$, then the chase with Σ terminates for instance I. □

Proof Sketch. It holds that Σ' contains all constraints that may fire during the execution of the chase starting with I and Σ. $I^{\Sigma'}$ is finite and $I^{\Sigma'} = I^{\Sigma}$. □

Hence, the crucial point is to effectively compute the set of (I, Σ)-irrelevant constraints. Unfortunately, it turns out that checking (I, Σ)-irrelevance is an undecidable problem in general:

THEOREM 5. Let Σ be a set of constraints, $\alpha \in \Sigma$ a constraint, and I an instance. It is undecidable if α is (I, Σ)-irrelevant. □

The proof of this theorem is given in the technical report [16]. This result prevents us from computing the minimal set of constraints that may fire when chasing I. Still, we can give sufficient conditions that guarantee (I, Σ)-irrelevance for a constraint. For this purpose, we use the chase graph.

PROPOSITION 6. Let I be an instance and Σ be a set of constraints. Further let $\alpha_I := \exists \overline{x} \bigwedge_{R(\overline{x}') \in I} R(\overline{x}')$ where $\overline{x} := \bigcup_{R(\overline{x}') \in I} \overline{x}'$. If the chase graph $G(\Sigma \cup \{\alpha_I\})$ contains no directed path from α_I to $\beta \in \Sigma$, then β is (I, Σ)-irrelevant. □

Proof Sketch Assume that β is not (I, Σ)-irrelevant. Then, there is a chase sequence $I \xrightarrow{\alpha_1, \overline{a_1}} I_1 \xrightarrow{\alpha_2, \overline{a_2}} \cdots \xrightarrow{\alpha_r, \overline{a_r}} I_r \xrightarrow{\beta, \overline{a}} \ldots$. If $\alpha_I \prec \beta$ we are finished. Otherwise, there must be some $n_r \in [r]$ such that $\alpha_{n_r} \prec \beta$ (otherwise β could not fire). If $\alpha_I \prec \alpha_{n_r}$ we are finished. Otherwise, there must be some $n_{r-1} \in [n_r - 1]$ such that $\alpha_{n_{r-1}} \prec \alpha_{n_r}$ (otherwise α_{n_r} could not fire). After some finite amount of iterations of this process we have that $\alpha_I \prec \alpha_{n_1} \prec \ldots \prec \alpha_{n_r} \prec \beta$. Therefore, the chase graph contains a directed path from α_I to β. □

Proposition 6 together with Lemma 2 gives us a sufficient data-dependent condition for chase termination, as illustrated in the following example.

EXAMPLE 12. Consider constraint set Σ from Figure 7 and q_2 from the beginning of this section. We set

$$\alpha_I := \exists c_1, x_1, x_2, y_1, y_2\, \mathit{rail}(c_1, x_1, y_1), \mathit{fly}(x_1, x_2, y_2), \mathit{fly}(x_2, x_1, y_2), \mathit{rail}(x_1, c_1, y_1)$$

and compute the chase graph

$$G(\Sigma \cup \{\alpha_I\}) := (\Sigma \cup \{\alpha_I\}, \{(\alpha_I, \alpha_1), (\alpha_3, \alpha_3)\}).$$

By Proposition 6, α_2 and α_3 are (I, Σ)-irrelevant. It holds that $\Sigma \setminus \{\alpha_2, \alpha_3\} = \{\alpha_1\}$ is inductively restricted, so we know from Lemma 2 that the chase of q_2 with Σ terminates. Similar argumentations hold for q_2'' and q_2''' from the beginning of Section 4. □

4.2 Monitoring Chase Execution

If the previous data-dependent termination condition does not apply, we propose to monitor the chase run and abort if tuples are created that may potentially lead to non-termination, an approach that is dynamical by nature. We introduce a data structure called *monitor graph*, which allows us to track the chase run.

DEFINITION 15. A *monitor graph* is a tuple (V, E), where $V \subseteq \Delta_{null} \times 2^{\mathrm{pos}(\Sigma)}$ and $E \subseteq V \times \Sigma \times 2^{\mathrm{pos}(\Sigma)} \times V$. □

A node in a monitor graph is a tuple (n, π), where n is a null value and π the set of positions in which n was first created (e.g. as null value with the help of some TGD). An edge $(n_1, \pi_1, \varphi_i, \Pi, n_2, \pi_2)$ between (n_1, π_1), (n_2, π_2) is labeled with the constraint φ_i that created n_2 and the set of positions Π from the body of φ_i in which n_1 occurred when n_2 was created. The monitor graph is successively constructed while running the chase, according to the following definition.

DEFINITION 16. The monitor graph $G_{\mathcal{S}} := G_r$ w.r.t. $\mathcal{S} = I_0 \xrightarrow{\varphi_0, \overline{a}_0} \ldots \xrightarrow{\varphi_{r-1}, \overline{a}_{r-1}} I_r$ is a monitor graph that is inductively defined as follows

- $G_0 = (\emptyset, \emptyset)$ is the empty chase segment graph.
- If $i < r$ and φ_i is an EGD then $G_{i+1} := G_i$.
- If $i < r$ and φ_i is a TGD then G_{i+1} is obtained from $G_i = (E_i, V_i)$ as follows. If the chase step $I_i \xrightarrow{\varphi_i, \overline{a}_i} I_{i+1}$ does not introduce any new null values, then $G_{i+1} := G_i$. Otherwise, E_{i+1} is set as the union of E_i and all pairs (n, π), where n is a newly introduced null value and π the set of positions in which n occurs. $V_{i+1} := V_i \cup \{ (n_1, \pi_1, \varphi_i, \Pi, n_2, \pi_2) \mid (n_1, \pi_1) \in E_i, (n_2, \pi_2) \in E_{i+1} \setminus E_i$ and Π is the set of positions in $\mathit{body}(\varphi_i(\overline{a}_i))$ where n_1 occurs $\}$. □

The size of the monitor graph is polynomial in the length of the chase sequence plus the length of the constraints' encoding. We illustrate the definition of the chase graph by a small example.

EXAMPLE 13. Consider the constraint $\Sigma_3 = \{\alpha_3\}$, where $\alpha_3 := S(x_3), R_k(x_1, x_2, x_3) \rightarrow \exists y R_k(y, x_1, x_2)$ from Example 11. Assume we have an instance of the form $I_0 := \{S(a_1), S(a_2), S(a_3), E(a_1, a_2, a_3)\}$. Then, the only chase sequence is $I_0 \rightarrow I_1 \rightarrow I_2 \rightarrow I_3$, where $I_1 = I_0 \cup \{E(y_1, a_1, a_2)\}$, $I_2 = I_1 \cup \{E(y_2, y_1, a_1)\}$ $I_3 = I_2 \cup \{E(y_3, y_2, y_1\}$. As y_1 is not in relation S the chase terminates. The monitor graph contains the path $(y_1, \{E^1\}) \overset{\alpha_3, E^1}{\longrightarrow} (y_2, \{E^1\}) \overset{\alpha_3, E^1}{\longrightarrow} (y_3, \{E^1\})$ plus an additional edge $(y_1, \{E^1\}) \overset{\alpha_3, E^2}{\longrightarrow} (y_3, \{E^1\})$. □

Our next task is to define a necessary criterion for non-termination on top of the monitor graph. To this end, we introduce the notion of *k-cyclicity*.

DEFINITION 17. Let $G = (V, E)$ be a monitor graph and $k \in \mathbb{N}$. G is called k-cyclic if and only if there are pairwise distinct edges $v_1, ..., v_k \in E$ such that

- there is a path in E that sequentially contains v_1 to v_k and
- for all $i \in [k-1]$: $p_{2,3,4,6}(v_i) = p_{2,3,4,6}(v_{i+1})$. □

EXAMPLE 14. Consider the scenario from Example 13. According to the previous definition, the chase graph presented there is 2-cyclic, but not 3-cyclic. □

We call a chase sequence *k-cyclic* if its monitor graph is k-cyclic. A chase sequence may potentially be infinite if some finite prefix is k-cyclic, for any $k \geq 1$:

LEMMA 3. Let $k \in \mathbb{N}$. If there is some infinite chase sequence $\mathcal{S}$ when chasing I_0 with Σ, then there is some finite prefix of $\mathcal{S}$ that is k-cyclic. □

To avoid non-termination, an application can fix a cycle-depth k and stop the chase when this limit is exceeded. For every terminating chase sequence there is a k such that the sequence is not k-cyclic, so if k is chosen large enough the chase will succeed. We argue that k-cyclicity is a *natural* condition that considers situations that may cause non-termination, so this approach is preferable to blindly chasing the instance and stopping after a fixed amount of chase steps. As justified by the following proposition, applications can choose k following a pay-as-you-go principle: for larger k-values the chase succeeds in more cases.

PROPOSITION 7. For $k \in \mathbb{N}$ there is Σ_k, I_k such that (a) both Σ_k and the subset of constraints in Σ_k that are not (I_k, Σ_k)-irrelevant are not inductively restricted; (b) every chase sequence for I_k with Σ_k is $(k-1)$-, but not k-cyclic. □

5. AN APPLICATION

Answering Conjunctive Queries on knowledge bases has recently gained attraction [5, 6]. Such knowledge bases typically have a set of constraints associated, which imply additional tuples that are not materialized in the knowledge base itself. An important problem is query answering on the implied knowledge base. If the chase with these constraints terminates, query answering can be done by answering it on the chased knowledge. However, if no termination guarantees for the chase can be made, more sophisticated techniques for query answering are required. This problem was first considered in [13] and then generalized in [5] and [6]. In this section we leverage the methods developed in Section 3, showing that they can be used to make the algorithms given in [5, 6] applicable to broader classes of constraints.

In [5] the class of so-called weakly guarded TGDs was introduced, which make query answering under constrained databases decidable. We first review this notion. Later, we will generalize weakly guarded TGDs with our methods.

DEFINITION 18. Let Σ be a set of TGDs. We call Σ *weakly guarded* if for every $\alpha \in \Sigma$ there exists $g_\alpha \in body(\alpha)$ such that for any $\pi \in aff(\Sigma) \cap pos(\alpha)$ and every variable x_π that occurs in π it holds that x_π occurs also in g_α. □

If Σ is weakly guarded, we abbreviate this by *WGTGD*(Σ). It was first shown in [5] that if *WGTGD*(Σ), then answering Conjunctive Queries on I^Σ is decidable for every database instance I, even though I^Σ may be infinite. Although not stated explicitly, it follows from the proof of Lemma 27 in [5] that the crucial property for decidability of query answering of WGTGDs is that in every chase step there is an atom in the body of the constraint under consideration that contains all labeled nulls. We state this observation more precisely in the following definition.

DEFINITION 19. Let $\mathcal{S}$ be a chase sequence starting with the instance I. $\mathcal{S}$ has the guarded null property if for every chase step $I' \overset{\alpha, \overline{a}}{\longrightarrow} I''$ in $\mathcal{S}$ there is an atom in $body(\alpha)(\overline{a})$ that contains every element from $(\overline{a} \cap \Delta_{null}) \backslash dom(I)$ that occurs in $head(\alpha)(\overline{a})$. □

With this definition at hand we can generalize Lemma 27 in [5] to the following version, which follows implicitly from the proof of Lemma 27 in [5]. We denote by $tw(I^\Sigma)$ the treewidth of I^Σ. A formal definition of treewidth is given in the technical report [16].

LEMMA 4. If all chase sequences w.r.t. Σ and I have the guarded null property, then $tw(I^\Sigma) \leq |dom(I)| + max\{\ ar(R) \mid R \in \mathcal{R}\ \}$. □

Straightforwardly, we obtain the following theorem that is obtained from a result in [7] and the observation that in case that all chase sequences have the guarded null property, then if $I^\Sigma \wedge Q$ and $I^\Sigma \wedge \neg Q$ are satisfiable, they have models of finite treewidth (because I^Σ has such a model).

THEOREM 6. There is an algorithm that, for every set of TGDs Σ, Conjunctive Query q and database instance I such that every chase sequence has the guarded null property, correctly computes $q(I^\Sigma)$. □

Unfortunately, it is not known if it is decidable if all chase sequences have the guarded null property (given Σ and I as input), which justifies the research regarding sufficient syntactic restrictions on the constraint set such that all chase sequences with this constraint have the guarded null property.

The notion of affected positions is a rough syntactic overestimation on where labeled nulls may occur in a constraint body during the execution of the chase. With the help of 2-restriction systems, we can improve this overestimation. The following definition states that every TGD α must have an atom in its body that contains all variables occurring in $f(\alpha)$, where f is the function from the constraint set's minimal restriction system (cf. Definition 10). Intuitively, $f(\alpha)$ defines the set of positions in which null values may occur during the execution of the chase.

DEFINITION 20. Let Σ be a set of TGDs and $G'(\Sigma) = (G', f)$ its minimal 2-restriction system. We call Σ *restrictedly guarded*

if for every $\alpha \in \Sigma$ there exists $g_\alpha \in body(\alpha)$ such that for any $\pi \in f(\alpha)$ and every variable x_π that occurs in π it holds that x_π occurs also in g_α. □

We call g_α a restricted guard and write *RGTGD*(Σ) to denote that constraint set Σ is restrictedly guarded.

EXAMPLE 15. Consider the set of constraints $\Sigma := \{\alpha_1, \alpha_2, \alpha_3\}$, where $\alpha_1 := R(x_1, x_2), S(x_1, x_2) \rightarrow \exists y S(x_2, y)$, $\alpha_2 := S(x_1, x_2), S(x_3, x_1) \rightarrow R(x_2, x_1)$ and $\alpha_3 := T(x_1, x_2) \rightarrow \exists y S(y, x_2)$. The set of affected positions is *aff*(Σ) $= \{S^1, S^2, R^1, R^2\}$ and therefore α_2 violates the condition for weak guardedness because there is no atom that contains x_1, x_2, x_3. However, the constraint set is restrictedly guarded. The minimal 2-restriction system $((\Sigma, E), f)$ contains only an edge $E(\alpha_1, \alpha_2)$ (and no other edges) and we have that $f(\alpha_1) = f(\alpha_3) := \emptyset$ and $f(\alpha_2) := \{S^2\}$. The body of α contains the atom $S(x_1, x_2)$ which serves as its restricted guard. □

Next, we relate restricted guardedness to weak guardedness and also show the crucial property that restricted guardedness ensures the guarded null property.

LEMMA 5. Let Σ be a set of TGDs.

- *WGTGD*(Σ) implies *RGTGD*(Σ).
- There is some Σ s.t. *RGTGD*(Σ), but not *WGTGD*(Σ).
- For every database I it holds that if *RGTGD*(Σ), then every chase sequence with Σ and I has the guarded null property. □

Proof Sketch. Let (G', f) be the minimal 2-restriction system for Σ. We can show by induction on the number of steps needed to compute it that $\bigcup_{\alpha \in \Sigma} f(\alpha) \subseteq$ *aff*(Σ). This implies bullet one. Bullet two is proven by Example 15. Bullet three follows from the observation that if a constraint β is violated during the execution of the chase, say $J \not\models \beta(\overline{a})$, then every $(\overline{a} \cap \Delta_{null}) \backslash dom(I)$ appears in some position g^i_β of some restricted guard g_β in the body of β. From the construction of the minimal 2-restriction system it follows that $g^i_\beta \in f(\beta)$. □

As our final result, Lemma 5 and Theorem 6 imply:

COROLLARY 1. There is an algorithm that, for every *RGTGD*(Σ), Conjunctive Query q, and database instance I, correctly computes $q(I^\Sigma)$. □

6. CONCLUSIONS

We studied the termination of the well-known chase algorithm. By the best of our knowledge, this was the first study that – in addition to the constraints – takes the specific instance (respectively query) into account. As another major 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, such as [8, 13, 4, 12, 15, 2, 21, 1, 19], and also to the so-called core-chase presented in [9]. As a sample application, we applied our novel concepts in the context of [5], showing that they can be used to identify a larger set of TGDs for which the methods in that paper apply.

There are some interesting open questions left. First, it is unknown if the membership test for $T[k]$, which has been shown to be in coNP, is also coNP-complete. Second, it is left open if $\bigcup_{k \geq 2} T[k]$ is still decidable. Finally, it is an interesting question if the positive results on core computation in data exchange settings from [11] extend to the T-hierarchy.

7. REFERENCES

[1] A. Deutsch, L. Popa and V. Tannen. Query Reformulation with Constraints. *SIGMOD Record*, 35(1):65–73, 2006.

[2] A. Fuxman, P.G. Kolaitis, R.J. Miller and W. Tan. 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] C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *J. ACM*, 31(4):718–741, 1984.

[5] A. Calì, G. Gottlob, and M. Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In *Description Logics*, volume 353, 2008.

[6] A. Calì, G. Gottlob, and T. Lukasiewicz. Datalog+-: A Unified Approach to Ontologies and Integrity Constraints. In *ICDT*, pages 14–30, 2009.

[7] B. Courcelle. The Monadic Second-order Logic of Graphs II: Infinite Graphs of Bounded Width. *Mathematical Systems Theory*, 21(4):187–221, 1989.

[8] D. Maier, A. Mendelzon and Y. Sagiv. Testing Implications of Data Dependencies. In *SIGMOD*, pages 152–152, 1979.

[9] A. Deutsch, A. Nash, and J. Remmel. The Chase Revisited. In *PODS*, pages 149–158, 2008.

[10] A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints. In *ICDT*, pages 225–241, 2003.

[11] G. Gottlob and A. Nash. Efficient Core Computation in Data Exchange. *J. ACM*, 55(2), 2008.

[12] A. Y. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, 10(4):270–294, 2001.

[13] D. S. Johnson and A. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. In *PODS*, pages 164–169, 1982.

[14] J. J. King. QUIST: A System for Semantic Query Optimization in Relational Databases. In *VLDB*, pages 510–517, 1981.

[15] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[16] M. Meier, M. Schmidt, and G. Lausen. On Chase Termination Beyond Stratification, Technical Report. *CoRR*, abs/0906.4228, 2009.

[17] M. Meier, M. Schmidt, and G. Lausen. Stop the Chase, Extended Abstract. *Proc. Alberto Mendelzon International Workshop on Foundations of Data Management*, 2009.

[18] M. Meier, M. Schmidt, and G. Lausen. Stop the Chase, Technical Report. *CoRR*, abs/0901.3984, 2009.

[19] D. Olteanu, J. Huang, and C. Koch. SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases. In *ICDE*, pages 640–651, 2009.

[20] L. Popa and V. Tannen. An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. In *ICDT*, pages 39–57, 1999.

[21] R. Fagin, P.G. Kolaitis, R.J. Miller and L. Popa. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[22] M. Schmidt, M. Meier, and G. Lausen. Foundations of SPARQL Query Optimization, Technical Report. *CoRR*, abs/0812.3788, 2008.