Exploiting Equality Generating Dependencies in Checking Chase Termination

Marco Calautti
University of Calabria
87036 Rende, Italy
calautti@dimes.unical.it

Sergio Greco
University of Calabria
87036 Rende, Italy
greco@dimes.unical.it

Cristian Molinaro
University of Calabria
87036 Rende, Italy
cmolinaro@dimes.unical.it

Irina Trubitsyna
University of Calabria
87036 Rende, Italy
trubitsyna@dimes.unical.it

ABSTRACT

The chase is a well-known algorithm with a wide range of applications in data exchange, data cleaning, data integration, query optimization, and ontological reasoning. Since the chase evaluation might not terminate and it is undecidable whether it terminates, the problem of defining (decidable) sufficient conditions ensuring termination has received a great deal of interest in recent years. In this regard, several termination criteria have been proposed. One of the main weaknesses of current approaches is the limited analysis they perform on *equality generating dependencies* (EGDs).

In this paper, we propose sufficient conditions ensuring that a set of dependencies has at least one terminating chase sequence. We propose novel criteria which are able to perform a more accurate analysis of EGDs. Specifically, we propose a new stratification criterion and an adornment algorithm. The latter can both be used as a termination criterion and be combined with current techniques to make them more effective, in that strictly more sets of dependencies are identified. Our techniques identify sets of dependencies that are not recognized by any of the current criteria.

1. INTRODUCTION

The chase is a well-known algorithm originally proposed for classical database problems, such as query optimization, query containment and equivalence, dependency implication, and database schema design [4, 7, 24, 29]. In recent years, it has seen a revival of interest because of a wide range of applications where it plays a central role, such as data exchange, data cleaning and repairing, data integration, and ontological reasoning [15, 8, 6, 5, 11, 12, 19, 17].

The execution of the chase involves inserting tuples possibly with null values to satisfy *tuple generating dependencies* (TGDs), and replacing null values with constants or other null values to satisfy *equality generating dependencies* (EGDs). Specifically, the chase consists of applying a sequence of steps, where each step enforces a dependency that is not satisfied by the current instance. It might well be the case that multiple dependencies can be enforced and, in this case, the chase picks one nondeterministically. Different choices lead to different sequences, some of which might be terminating, while others might not. This aspect is illustrated in the following example.

Example 1. Consider the set of dependencies Σ_1 below:

$$r_1: \ N(x) \rightarrow \exists y \ E(x,y)$$
$$r_2: \ E(x,y) \rightarrow N(y)$$
$$r_3: \ E(x,y) \rightarrow x = y$$

and the database $D = \{N(a)\}$. All dependencies are satisfied by D, except for r_1. Thus, the chase enforces r_1 by adding $E(a, \eta_1)$ to D, where η_1 is a (labeled) null value. However, this causes both r_2 and r_3 to be violated: r_2 requires the fact $N(\eta_1)$, while r_3 says that a and η_1 should be the same. Suppose the chase chooses to enforce r_2, and thus $N(\eta_1)$ is added to the current instance. Now r_1 is not satisfied again, while r_3 continues to be violated. Suppose the chase chooses to enforce r_1. Then, similar to the first step, $E(\eta_1, \eta_2)$ is added to the current instance, and this causes r_2 to become violated again. It is easy to see that repeatedly enforcing first r_1 and then r_2 yields an infinite chase sequence that introduces an infinite number of facts: $N(\eta_2), E(\eta_2, \eta_3), N(\eta_3), \ldots$.

However, by enforcing first r_1 and then r_3, we get a terminating chase sequence. Specifically, enforcing r_1 adds $E(a, \eta_1)$ to D. Then, the application of r_3 updates the null value η_1 to a. At this point, no further dependency needs to be enforced, and the chase terminates with the resulting database being $\{N(a), E(a,a)\}$. □

The importance of the chase in many applications is due to the fact that several problems (e.g., checking query containment under dependencies, checking implication of dependencies, computing solutions in data exchange, and computing certain answers in data integration) can be solved by exhibiting a *universal model*, and the chase computes a universal model, when it terminates [13]. Roughy speaking, a *model* for a database and a set of dependencies is a finite instance that includes the database and satisfies the dependencies. A *universal model* is a model that can be "mapped" to every other model—in a sense, it represents the entire space of possible models (formal definitions are reported in Section 2). Universal models are slight generalizations of universal solutions in the data exchange setting [15], and can be used to compute them. Moreover, the certain answers to a conjunctive query in the presence of dependencies can be computed by evaluating the query over a universal model

Proceedings of the VLDB Endowment, Vol. 9, No. 5

(rather than considering all models). Other applications of universal models (e.g., dependency implication and query containment under dependencies) can be found in [13].

Thus, finding a universal model is a central problem in many applications and, once again, the chase is a tool to solve it, provided that it terminates. As a consequence, checking whether the chase terminates becomes a central problem, but unfortunately, it is an undecidable one [13, 22, 20]. To cope with this issue, several "termination criteria" have been proposed, that is, (decidable) sufficient conditions ensuring chase termination.

Indeed, as illustrated in Example 1 above, when we talk about chase termination, it is important to distinguish between two problems: checking whether *all* chase sequences are terminating, and checking whether there is *at least one* terminating chase sequence. Most of the work in the literature has focused on the problem of checking if all chase sequences are terminating (a thorough discussion of related work is reported in Section 3), independently of the considered database. However, since in many applications the ultimate goal is to compute a universal model, checking for the existence of a terminating chase sequence and constructing it suffices for the purpose.

In this regard, a universal model might be computed using the *core chase* [13], which is a variant of the standard chase where all applicable chase steps are fired "in parallel", rather than picking one non-deterministically as in the standard chase. One consequence of the parallel application is that nondeterminism is eliminated. Another important property of the core chase is that it is complete for finding universal models, that is, whenever a universal model exists, the core chase terminates and finds such a model. Thus, if we know that there exists a terminating standard chase sequence (and thus a universal model), then we can use the core chase to compute a universal model.

Furthermore, the weaker requirement of checking for the existence of a terminating chase sequence, rather than ensuring that every chase sequence is terminating, can be profitably leveraged to identify more sets of dependencies for which we can compute a universal model. For instance, the set of dependencies Σ_1 of Example 1 might be identified by a criterion ensuring termination of at least one chase sequence. However, every criterion requiring all chase sequences to be terminating will not recognize Σ_1, thereby providing no information about whether we can compute a universal model.

Despite the significant body of work in this area, there are still large classes of dependencies for which the chase is not applicable as termination cannot be statically established.

One weakness of current approaches is that the analysis of EGDs is limited or absent altogether. In fact, more general approaches, such as *super-weak acyclicity* [30], semi-dynamic approaches [23], and rewriting approaches [25, 26, 27], were meant to guarantee termination of TGDs only. Other approaches, such as *weak acyclicity* [15] and *safety* [32], guarantee the termination of a set of TGDs and EGDs, but do not analyze EGDs at all, which leads them to impose strong conditions on TGDs to guarantee termination. Firing relations among dependences used in stratification-based approaches [13, 32, 26] consider EGDs in a limited way. To mitigate the aforementioned issues, an "indirect" way of dealing with EGDs was proposed in [21, 30], where a set Σ of TGDs and EGDs is rewritten into a set Σ' containing only TGDs, and termination analysis is carried out on Σ'. The aim is to "simulate" the behavior of the EGDs by means of TGDs. While these preprocessing steps ensure soundness, i.e., if all chase sequences of Σ' are terminating then all chase sequences of Σ are terminating, they are not complete, i.e., the implication in the opposite direction does not hold.

Treating EGDs as first-class citizens is very important, as they are among the most popular classes of dependencies in real applications, playing a critical role in maintaining data integrity, query optimization and indexing, and schema design [14]. For instance, functional dependencies can be expressed by EGDs. In very simple scenarios, such as Example 1 above, current termination criteria are not able to say whether a universal model can be found. As a further scenario, Example 8 shows a simple set of dependencies for which all chase sequences are terminating, but there is no terminating chase sequence for the set of dependencies obtained from the EGD simulation.

In this paper, we propose new sufficient conditions ensuring that a set of dependencies (possibly containing both TGDs and EGDs) admits at least one terminating chase sequence, independently of the database. Our approach performs an explicit analysis of EGDs and identifies sets of dependencies that are not captured by any of the current techniques. To the best of our knowledge, sufficient conditions ensuring termination of at least one chase sequence was studied only in [31, 32].

Contributions. The main contributions of the paper are as follows.

- First of all, for the different variants of the chase, we study the relationships between the classes of sets of dependencies for which all chase sequences are terminating or at least one chase sequence is terminating, when sets of dependencies can contain also EGDs—previous work has addressed this problem in the presence of TGDs only.
- We propose a new stratification criterion ensuring the existence of at least one terminating chase sequence, which strictly generalizes stratification [13] and allows us to identify sets of dependencies not included by any of the current termination criteria.
- We then propose an adornment algorithm which is used to define sufficient conditions ensuring the existence of at least one terminating chase sequence, independently of the database. The algorithm performs a direct analysis of EGDs and exploits them to try to identify a terminating chase sequence. The aim of the algorithm is twofold: *(i)* it defines a termination criterion on its own, and *(ii)* it can be combined with other termination criteria to make them more effective, in that strictly more sets of dependencies can be identified by using our algorithm in conjunction with a termination criterion.
- To assess our approach, we carried out an experimental evaluation over 178 real-world datasets. The experimental results show that our technique is very effective (among 76 datasets for which the chase terminated, only 2 were not recognized) and also efficient (in most of the cases the algorithm's running time is lower than one second).

2. PRELIMINARIES

Basics. We assume the existence of the following pairwise disjoint sets of symbols: an infinite set *Consts* of *constants*,

an infinite set *Nulls* of *labeled nulls*, and an infinite set *Vars* of *variables*. A *term* is a constant, a labeled null, or a variable. A *schema* is a finite set $\mathbf{R}$ of *predicates*, where each predicate R is associated with an arity $ar(R)$, which is a non-negative integer.

An *atom* over $\mathbf{R}$ is an expression of the form $R(t_1, \ldots, t_n)$, where R is an n-ary predicate in $\mathbf{R}$ and each t_i is a term—we denote an atom also as $R(\mathbf{t})$, where $\mathbf{t}$ is understood to be a sequence of n terms. If $t_i \in Consts \cup Nulls$ for every $1 \leq i \leq n$, then the atom is also called a *fact*.

Given a set of atoms A, we use $Consts(A)$ (resp. $Nulls(A)$, $Vars(A)$) to denote the set of all constants (resp. labeled nulls, variables) occurring in A, and use $Dom(A)$ to denote the set $Consts(A) \cup Nulls(A) \cup Vars(A)$.

An *instance* over $\mathbf{R}$ is a set of facts over $\mathbf{R}$, while a *database* is an instance where only constants appear. We will use D (resp. I, J, and K), possibly subscripted or primed, to refer to databases (resp. instances).

A *tuple generating dependency* (TGD) over $\mathbf{R}$ is a formula r of the form:

$$\forall \mathbf{x} \forall \mathbf{y}\ \varphi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x}, \mathbf{z})$$

where $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are lists of variables, and $\varphi(\mathbf{x}, \mathbf{z})$ (resp. $\psi(\mathbf{x}, \mathbf{y})$) is a conjunction of atoms over $\mathbf{R}$ whose variables are exactly $\mathbf{x}$ and $\mathbf{y}$ (resp. $\mathbf{x}$ and $\mathbf{z}$) and is called the *body* (resp. *head*) of r, denoted as $Body(r)$ (resp. $Head(r)$). With a slight abuse of notation, we sometimes treat $Body(r)$ and $Head(r)$ as sets (of atoms). A TGD is said to be *universally quantified* or *full* if all its variables are universally quantified (i.e., $\mathbf{z}$ is empty), otherwise it is *existentially quantified*.

An *equality generating dependency* (EGD) over $\mathbf{R}$ is a (universally quantified) formula of the form:

$$\forall \mathbf{x} \forall \mathbf{y}\ \varphi(\mathbf{x}, \mathbf{y}) \rightarrow x_1 = x_2$$

where $\mathbf{x} = x_1, x_2$, $\mathbf{y}$ is a list of variables, $\varphi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms over $\mathbf{R}$ whose variables are exactly $\mathbf{x}$ and $\mathbf{y}$.

In the following, we will omit the universal quantification in front of dependencies and assume that all variables appearing in the body are universally quantified. Labeled nulls are not allowed to occur in dependencies. Throughout this paper we assume we are given an arbitrary but fixed schema $\mathbf{R}$. Unless otherwise stated, an atom (database, instance, dependency, etc.) is understood to be over $\mathbf{R}$. In Example 1 above, r_1 and r_2 are TGDs, with r_1 being existentially quantified and r_2 being full, while r_3 is an EGD.

A *homomorphism* from a set of atoms A_1 to a set of atoms A_2 is a mapping $h : Dom(A_1) \rightarrow Dom(A_2)$ such that:

- $h(c) = c$, for every $c \in Consts(A_1)$; and
- for every atom $R(t_1, \ldots, t_n)$ in A_1, we have that $R(h(t_1), \ldots, h(t_n))$ is in A_2.

With a slight abuse of notation, we apply h also to sets of atoms and thus, for a given set of atoms A, we define $h(A) = \{R(h(t_1), \ldots, h(t_n)) \mid R(t_1, \ldots, t_n) \in A\}$.

Example 2. Consider the database $D = K_1 = \{N(a)\}$ and the set of dependencies Σ_1 of Example 1.

Let $h_1 : Dom(Body(r_1)) \rightarrow Dom(K_1)$ be defined as follows: $h_1(x) = a$. Clearly, h_1 is a homomorphism from the body of r_1 to K_1. Consider now the instance $K_2 = \{N(a), E(a, \eta_1)\}$, and let h_2 be the mapping defined as follows: $h_2(x) = a$ and $h_2(y) = \eta_1$. It is easy to see that h_2 is a homomorphism from the body of r_2 to K_2. Moreover, h_2 is also a homomorphism from the body of r_3 to K_2. □

Universal models. Given a database D and a set of dependencies Σ, a *model* of (D, Σ) is a finite instance J such that $D \subseteq J$ and $J \models \Sigma$ (i.e., J satisfies all dependencies in Σ in the standard first-order manner). A *universal model* of (D, Σ) is a model J of (D, Σ) such that for every model J' of (D, Σ) there exists a homomorphism from J to J'. The set of all models (resp. universal models) of (D, Σ) will be denoted by $Mod(D, \Sigma)$ (resp. $UMod(D, \Sigma)$).

Example 3. Consider the set of dependencies Σ_3 below:

$$r_1 :\ P(x, y) \rightarrow \exists z\ E(x, z)$$
$$r_2 :\ Q(x, y) \rightarrow \exists z\ E(z, y)$$

and the database $D = \{P(a, b), Q(c, d)\}$. Both $J_1 = D \cup \{E(a, \eta_1), E(\eta_2, d)\}$ and $J_2 = D \cup \{E(a, d)\}$ are models of (D, Σ_3). It can be shown that J_1 is a universal model, while J_2 is not. Notice that an homomorphism from J_1 to J_2 is the mapping h defined as follows: $h(\eta_1) = d$ and $h(\eta_2) = a$. In a sense, J_2 makes the somehow arbitrary assumption that the two facts required by the two TGDs are the same fact $E(a, d)$, which is not part of the specification. □

As discussed below, computing certain query answers is one of many applications where universal models play an important role, and their computation is a central problem. Consider an instance J and a query Q. Then, *(i)* $J_{\downarrow}$ denotes the set of facts in J that do not contain labeled nulls, and *(ii)* $Q(J)$ denotes the result of evaluating Q over J.

The certain answers to a query Q over a database D and a set of dependencies Σ are defined as $certain(Q, D, \Sigma) = \bigcap\{Q(J) \mid J \in Mod(D, \Sigma)\}$. The certain answers to a union of conjunctive queries Q can be computed by evaluating Q over an arbitrary universal model, that is, $certain(Q, D, \Sigma) = Q(I)_{\downarrow}$, where $I \in UMod(D, \Sigma)$. This means that to determine the certain answers to a union of conjunctive queries Q over a database D with dependencies Σ, it is not necessary to compute all models of (D, Σ), but it suffices to compute just an arbitrary universal model. Therefore, the computation of a universal model is particularly relevant. It is worth mentioning that the aforementioned property has applications in query answering under dependencies, query answering in data exchange, and query answering with incomplete and inconsistent data [15, 10].

The chase. The *chase* takes as input a database D and a set Σ of dependencies, and whenever it terminates without failing, it constructs a universal model of (D, Σ) [13, 15].

Below we define a *chase step*, which consists of enforcing a TGD or an EGD. As detailed later, the chase step is used by different variants of the chase (standard, oblivious, semi-oblivious), each of which relies on a different condition of "applicability" of the chase step. Thus, the following definition does not incorporate a notion of applicability, but it will be combined with different notions of applicability to define the different variants of the chase.

A *substitution* γ is either the empty set or a singleton $\{\eta/t\}$, where η is a labeled null and t is either a labeled null or a constant. The result of applying γ to an expression F (e.g., term, atom, set of atoms, etc.), denoted $F\gamma$, is F if $\gamma = \emptyset$, otherwise it is the expression obtained from F by replacing every occurrence of η with t.

Definition 1 (Chase step). Let K be an instance, r a dependency, and h a homomorphism from $Body(r)$ to K.

An expression of the form $K \xrightarrow{r,h,\gamma} J$ is a *chase step* if the following conditions hold.

1. If r is a TGD $\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x},\mathbf{z})$ then let h' be the homomorphism obtained by extending h so that each variable in $\mathbf{z}$ is assigned a fresh labeled null not occurring in K. Then, $J = K \cup h'(\psi(\mathbf{x},\mathbf{z}))$. Furthermore, γ is the empty substitution.
2. If r is an EGD $\varphi(\mathbf{x},\mathbf{y}) \rightarrow x_1 = x_2$ then $h(x_1) \neq h(x_2)$. Furthermore,
   (a) If $h(x_1), h(x_2) \in \mathit{Consts}$, then $J = \bot$ and γ is the empty substitution.
   (b) Otherwise, γ and J are defined as follows. If $h(x_1)$ is a labeled null, then $\gamma = \{h(x_1)/h(x_2)\}$; otherwise, $\gamma = \{h(x_2)/h(x_1)\}$. Moreover, $J = K\,\gamma$.

In a chase step, γ is used to keep track of the substitution performed when an EGD is enforced.

Example 4. Consider again the database $D = K_1 = \{N(a)\}$ and the set of dependencies Σ_1 of Example 1. Let h_1 be the homomorphism of Example 2. Then, $K_1 \xrightarrow{r_1,h_1,\gamma_1} K_2$ is a chase step, where $K_2 = K_1 \cup \{E(a,\eta_1)\} = \{N(a), E(a,\eta_1)\}$ and γ_1 is the empty substitution (as r_1 is a TGD). Consider now the homomorphism h_2 of Example 2. Then, $K_2 \xrightarrow{r_2,h_2,\gamma_2} K_3$ is a chase step, where $K_3 = K_2 \cup \{N(\eta_1)\} = \{N(a), E(a,\eta_1), N(\eta_1)\}$ and γ_3 is the empty substitution. Another possible chase step starting from K_2 is $K_2 \xrightarrow{r_3,h_2,\gamma_2'} K_3'$, where $\gamma_2' = \{\eta_1/a\}$ and $K_3' = K_2\,\gamma_2' = \{N(a), E(a,a)\}$. □

A *chase sequence* of (D,Σ) is a (possibly infinite) sequence of chase steps $S = K_1 \xrightarrow{r_1,h_1,\gamma_1} K_2 \xrightarrow{r_2,h_2,\gamma_2} K_3 \cdots$ such that $K_1 = D$ and every $r_i \in \Sigma$. Moreover:

- S is a *standard chase sequence* if it is an exhaustive application of chase steps s.t. for each $K_i \xrightarrow{r_i,h_i,\gamma_i} K_{i+1}$ in S, if r_i is a TGD, then there is no extension of h_i to a homomorphism h_i' from $\mathit{Body}(r_i) \cup \mathit{Head}(r_i)$ to K_i.
- S is an *oblivious chase sequence* if it is an exhaustive application of chase steps s.t. for each $K_i \xrightarrow{r_i,h_i,\gamma_i} K_{i+1}$ in S, there is no chase step $K_j \xrightarrow{r_j,h_j,\gamma_j} K_{j+1}$ in S such that $j < i$, $r_j = r_i = r$, and for each variable x occurring in the body of r we have that $h_i(x) = h_j(x)\,\gamma_j \cdots \gamma_{i-1}$.
- S is a *semi-oblivious chase sequence* if it is an exhaustive application of chase steps s.t. for each $K_i \xrightarrow{r_i,h_i,\gamma_i} K_{i+1}$ in S, there is no chase step $K_j \xrightarrow{r_j,h_j,\gamma_j} K_{j+1}$ in S such that $j < i$, $r_j = r_i = r$, and for each variable x occurring in both the body and the head of r we have that $h_i(x) = h_j(x)\,\gamma_j \cdots \gamma_{i-1}$.

Example 5. Consider again the database $D = \{N(a)\}$ and the set of dependencies Σ_1 of Example 1. A standard chase sequence of D with Σ_1 is $K_1 \xrightarrow{r_1,h_1,\gamma_1} K_2 \xrightarrow{r_3,h_2,\gamma_2'} K_3'$, where $K_1 = D$ and $h_1, h_2, \gamma_1, \gamma_2', K_2, K_3'$ are those reported in Example 4. Notice that no further chase steps can be added to the sequence.

As mentioned in Example 1, another standard chase sequence of D with Σ_1 is the (infinite) one obtained by repeatedly enforcing first r_1 and then r_2, that is $K_1 \xrightarrow{r_1,h_1,\gamma_1} K_2 \xrightarrow{r_2,h_2,\gamma_2} K_3 \ldots$, where $h_1, h_2, \gamma_1, \gamma_2, K_2$, and K_3 are those reported in Example 4. □

The following example shows the different behaviors of standard, oblivious, and semi-oblivious chase sequences.

Example 6. Consider the database $D = K_1 = \{E(a,b)\}$ and a set Σ_6 consisting only of the following TGD r:

$$E(x,y) \rightarrow \exists z\ E(x,z)$$

Since $D \models r$, the only standard chase sequence of D with Σ is the empty sequence.

A non-empty (terminating) semi-oblivious chase sequence is $K_1 \xrightarrow{r,h_1,\gamma_1} K_2$, where $h_1(x) = a$, $h_1(y) = b$, γ_1 is the empty substitution, and $K_2 = K_1 \cup \{E(a,\eta_1)\} = \{E(a,b), E(a,\eta_1)\}$. Notice that adding the chase step $K_2 \xrightarrow{r,h_2,\gamma_2} K_3$, with $h_2(x) = a$, $h_2(y) = \eta_1$, $\gamma_2 = \emptyset$, and $K_3 = K_2 \cup \{E(a,\eta_2)\}$, does not result in a semi-oblivious chase sequence, because of the presence of the chase step $K_1 \xrightarrow{r,h_1,\gamma_1} K_2$ in the same chase sequence, with $h_1(x)\gamma_1 = h_2(x) = a$.

As for the oblivious chase, the infinite sequence whose first step is $K_1 \xrightarrow{r,h_1,\gamma_1} K_2$ discussed above, and the i-th chase step ($i > 1$) is $K_i \xrightarrow{r,h_i,\gamma_i} K_{i+1}$, with $h_i(x) = a$, $h_i(y) = \eta_{i-1}$, $\gamma_i = \emptyset$, and $K_{i+1} = K_i \cup \{E(a,\eta_i)\}$ is an (infinite) oblivious chase sequence. □

A standard (resp. oblivious, semi-oblivious) chase sequence S can be finite (when no further chase step can be applied) or infinite (when there is always a further chase step that can be applied)—in the former case we also say that the sequence is *terminating*. If S is finite and consists of m chase steps, we say that K_m is the *result* of S. If $K_m = \bot$ then S is *failing*, otherwise it is *successful*. For instance, the first standard chase sequence discussed in Example 5 is terminating, successful, and its result is K_3'. The second standard chase sequence in Example 5 is not terminating.

In the presence of TGDs only, the oblivious (resp. semi-oblivious) chase procedure is equivalent to the computation of the fixpoint of a particular Skolemized version of Σ with D, where Skolemized terms are used in place of labeled nulls. For instance, the Skolemized version of dependency r in Example 6 for the oblivious (resp., semi-oblivious) chase is $E(x,y) \rightarrow E(x, f_z^r(x,y))$ (resp., $E(x,y) \rightarrow E(x, f_z^r(x))$).

As shown in [15], for every database D and set of dependencies Σ, (1) if J is the result of some successful terminating standard chase sequence of D with Σ, then J is a universal model of (D,Σ), called *canonical*; (2) if some failing standard chase sequence of D with Σ exists, then there is no model of (D,Σ). We use $\mathit{CMod}(D,\Sigma)$ to denote the set of all canonical models of (D,Σ). In some cases, we cannot produce a universal model by the chase as there is no terminating sequence, although a model does exist.

The *core chase* has been proposed to identify a preferable universal model [13, 16]. To define the core chase, we first need to introduce the notion of a core of an instance. Roughly speaking, the core of an instance J is the smallest subset of J that is also a homomorphic image of J. More precisely, a subset C of an instance J is a *core of* J if there is a homomorphism from J to C, but there is no homomorphism from J to a proper subset of C. Cores of J are unique up to isomorphism and therefore we can talk about "the" core of J, which is denoted as $\mathit{core}(J)$.

A *core chase sequence* is a sequence of *core chase steps*. Roughly speaking, a core chase step first applies all possible standard chase steps "in parallel", and then computes the

core of the resulting instance. As all standard chase steps are applied in parallel, the core chase eliminates the nondeterminism of the standard chase. More formally, given an instance K and a set of dependencies Σ, a core chase step consists of the following two sub-steps: *(i)* $J = \cup_{K \xrightarrow{r,h,\gamma} K'} K'$, where each $K \xrightarrow{r,h,\gamma} K'$ is understood to be a standard chase step; *(ii)* $J' = core(J)$. Then, J' is the result of the core chase step. [13] showed that whenever there is a universal model of (D, Σ), the core chase is able to construct one, that is, the core chase is a *complete* procedure for finding universal models. Moreover, every core chase sequence of D with Σ constructs the same (up to isomorphism) universal model.

EXAMPLE 7. Consider the database D and the set of dependencies $\Sigma_6 = \{r\}$ of Example 6. Recall that there is no standard chase step involving D and r. As the core chase starts by applying all standard chase steps, the only core chase sequence is the empty one, similar to the standard chase case. □

In the following, whenever a successful terminating c-chase sequence of D with Σ does exist, where $\mathsf{c} \in \{\mathsf{std}, \mathsf{obl}, \mathsf{sobl}, \mathsf{core}\}$ stands for the standard, oblivious, semi-oblivious, and core chase, respectively, we use $chase^{\mathsf{c}}(D, \Sigma)$ to denote one of the homomorphically equivalent universal models constructed by the c-chase. If there is a failing c-chase sequence of D with Σ, we write $chase^{\mathsf{c}}(D, \Sigma) = \bot$.

Termination Classes. We denote by $\mathsf{CT}^{\mathsf{c}}_{\forall}$, with $\mathsf{c} \in \{\mathsf{std}, \mathsf{obl}, \mathsf{sobl}, \mathsf{core}\}$, the class of sets of dependencies Σ such that for every database D all c-chase sequences of D with Σ are terminating. Analogously, we denote by $\mathsf{CT}^{\mathsf{c}}_{\exists}$ the class of sets of dependencies Σ such that for every database D there is a terminating c-chase sequence of D with Σ.

Even focusing on TGDs only, the problem of verifying whether a set of dependencies belongs to $\mathsf{CT}^{\mathsf{c}}_{\forall}$ or $\mathsf{CT}^{\mathsf{c}}_{\exists}$, for $\mathsf{c} \in \{\mathsf{std}, \mathsf{obl}, \mathsf{sobl}, \mathsf{core}\}$, is undecidable [20, 22]. Thus, the best practical approach is to find relevant decidable classes of dependencies included in these classes. For sets of TGDs only, it has already been shown in [31, 33] that:

$$\mathsf{CT}^{\mathsf{obl}}_{\forall} = \mathsf{CT}^{\mathsf{obl}}_{\exists} \subsetneq \mathsf{CT}^{\mathsf{sobl}}_{\forall} = \mathsf{CT}^{\mathsf{sobl}}_{\exists} \subsetneq \mathsf{CT}^{\mathsf{std}}_{\forall} \subsetneq \mathsf{CT}^{\mathsf{std}}_{\exists} \subsetneq \mathsf{CT}^{\mathsf{core}}_{\forall} = \mathsf{CT}^{\mathsf{core}}_{\exists}$$

The above hierarchy is relevant because if we determine that a set of TGDs belongs to $\mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$ with $\mathsf{q} \in \{\forall, \exists\}$ and $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}\}$, then Σ belongs to $\mathsf{CT}^{\mathsf{std}}_{\forall}$ (and, of course, $\mathsf{CT}^{\mathsf{std}}_{\exists}$), and, in some cases, the analysis of the oblivious or semi-oblivious chase is easier. In fact, the importance of these chase variants has been widely recognized and their behavior has been studied in different works [9, 23, 27, 30, 31].

In this paper, we introduce new (decidable) sufficient conditions for a set of dependencies to be in $\mathsf{CT}^{\mathsf{std}}_{\exists}$.

3. RELATED WORK

As mentioned in the introduction, several sufficient conditions for chase termination have been proposed over the years—we call them *termination criteria*.
We will use calligraphic style $\mathcal{C}$ to denote the class of sets of dependencies recognized by a criterion C (written in italics).

Static approaches. The first and basic effort concerning the formalization of a (decidable) sufficient condition guaranteeing that all standard chase sequences are terminating, independently from the database, is *weak acyclicity* (*WA*) [15]. Roughly speaking, it checks whether the TGDs do not allow for nulls to cyclically propagate. The approach works for sets of dependencies containing both TGDs and EGDs, even though the latter are ignored in the analysis (as a strong condition is imposed on TGDs).

An extension of weak acyclicity, called *stratification* (*Str*), has been proposed in [13]. The idea behind stratification is to decompose the set of dependencies into independent subsets, where each subset consists of dependencies that may fire each other, and to check each component separately for weak acyclicity. However, [31] showed that stratification is not able to check whether all standard chase sequences are terminating (as weak acyclicity does), but ensures only that there is a terminating standard chase sequence. A variant of stratification, called *c-stratification* (*CStr*), guaranteeing that all standard chase sequences are terminating, has been proposed in [31]. C-stratification is defined in the same way as stratification, but the oblivious chase is used instead of the standard one to determine whether a dependency fires another. Both *Str* and *CStr* allow TGDs as well as EGDs, but the analysis of EGDs is limited to the firing relation only. A different extension of weak acyclicity, called *safety* (*SC*), has been proposed in [32]. The improvement is obtained by considering only "affected" positions [10], that is, positions which may actually contain null values. The approach works for sets of dependencies containing both TGDs and EGDs, but the latter are neglected altogether in the analysis.

Another extension of weak acyclicity (which indeed strictly extends *SC*) has been introduced in [30] under the name of *super-weak acyclicity* (*SwA*). In addition to considering how dependencies may activate each other, *SwA* also takes into account the fact that the same variable may appear more than once in the body, and thus a dependency is not fired when different nulls are inserted in positions associated with the same variable. The analysis is carried out by using the semi-oblivious chase. The approach is defined for sets of TGDs only, as EGDs are emulated via "substitution-free simulation", which will be discussed in Section 4.

Safe restriction (*SR*) and *inductive restriction* (*IR*) extend c-stratification, but still perform a limited analysis of EGDs [32]. In terms of expressivity, these approaches are not comparable with *SwA*. Both *SwA* and *IR* have been extended by the *Local Stratification* (*LS*) criterion [26]; however, *LS* neglects EGDs altogether.

As for the relative expressivity of the termination criteria discussed above, [27] showed that $\mathcal{CS}tr \subsetneq \mathcal{SR} \subsetneq \mathcal{IR} \subsetneq \mathcal{LS}$ and $\mathcal{SwA} \subsetneq \mathcal{LS}$.

Semi-dynamic approaches. In [23], the *model-faithful acyclicity* (MFA) and *model-summarising acyclicity* (MSA) techniques have been proposed. The idea is to run the oblivious (or semi-oblivious) chase and then use sufficient checks to identify cyclic computations. Since no sufficient, necessary, and computable test can be given for the latter, [23] adopted an approach of "raising the alarm" and stop the process if a "cyclic" term $f(t)$ is derived, i.e., where f occurs in t. This is done in a declarative way by extending a given set of dependencies Σ into a new set Σ', and then checking whether Σ' does not entail a special predicate. The two aforementioned techniques are defined for TGDs only, as EGDs are assumed to be emulated through substitution-free simulation (discussed in Section 4).

Rewriting approaches. Rewriting techniques for checking chase termination have been proposed in [25, 26, 27].

They consist in rewriting a set of TGDs Σ into a new set Σ^α with the aim of verifying structural properties for chase termination on Σ^α rather than Σ. These techniques have been defined for TGDs only and perform an analysis of the semi-oblivious chase. [27] showed that most of the termination criteria improve if we consider adorned TGDs rather than the original ones. The rewriting approach has also been used to define the *acyclicity* (*AC*) criterion.

The termination criteria discussed in this section ensure that all standard chase sequences are terminating, except for stratification (which ensures the existence of at least one terminating standard chase sequence), and perform a limited analysis of EGDs (or no analysis altogether), thereby imposing stronger conditions on TGDs. In contrast, the criteria proposed in this paper ensure that at least one standard chase sequence is terminating. It is worth noticing that constructing one terminating standard chase sequence suffices for the purpose of getting a universal model. By considering this weaker condition to be ensured and by performing a direct analysis of EGDs, our techniques identify sets of dependencies that are not captured by any of the aforementioned criteria.

4. DEALING WITH EGDs

Before presenting our criteria (in Sections 5 and 6), we shed light on the relationships between the classes $\mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$, where $\mathsf{q} \in \{\forall, \exists\}$ and $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}, \mathsf{core}\}$, when arbitrary sets of dependencies are considered. Recall that a hierarchy for sets consisting only of TGDs has been presented in Section 2, but the relationships in the presence of both TGDs and EGDs have not been studied so far (to the best of our knowledge). We also discuss different issues arising in the presence of EGDs.

In the rest of the paper, given two sets C_1 and C_2, we write $C_1 \not\parallel C_2$ iff $C_1 \not\subseteq C_2$ and $C_2 \not\subseteq C_1$.

THEOREM 1. *For general dependencies (including TGDs and EGDs), the following relations hold:*

1. $\mathsf{CT}^{\mathsf{c}}_{\forall} \subsetneq \mathsf{CT}^{\mathsf{c}}_{\exists}$ *for* $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$, *and* $\mathsf{CT}^{\mathsf{core}}_{\forall} = \mathsf{CT}^{\mathsf{core}}_{\exists}$;
2. $\mathsf{CT}^{\mathsf{obl}}_{\mathsf{q}} \subsetneq \mathsf{CT}^{\mathsf{sobl}}_{\mathsf{q}} \subsetneq \mathsf{CT}^{\mathsf{std}}_{\mathsf{q}} \subsetneq \mathsf{CT}^{\mathsf{core}}_{\mathsf{q}}$ *for* $\mathsf{q} \in \{\forall, \exists\}$;
3. $\mathsf{CT}^{\mathsf{obl}}_{\exists} \not\parallel \mathsf{CT}^{\mathsf{sobl}}_{\forall}$, $\mathsf{CT}^{\mathsf{sobl}}_{\exists} \not\parallel \mathsf{CT}^{\mathsf{std}}_{\forall}$, *and* $\mathsf{CT}^{\mathsf{obl}}_{\exists} \not\parallel \mathsf{CT}^{\mathsf{std}}_{\forall}$. □

The relationships between the classes $\mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$, where $\mathsf{q} \in \{\forall, \exists\}$ and $\mathsf{c} \in \{\mathsf{std}, \mathsf{obl}, \mathsf{sobl}, \mathsf{core}\}$, are shown in Table 1, for the case of TGDs only (such results are known) and in the presence of both TGDs and EGDs (shown in this paper).

As discussed in the previous section, chase termination criteria proposed in the literature focus on TGDs considering EGDs in a very limited way. More general approaches (including *SwA*, *LS*, *MFA*, *MSA*) as well as rewriting techniques were meant to guarantee termination of TGDs only.

An "indirect" way of dealing with EGDs has been proposed in [21, 30]. Specifically, the analysis of a set of dependencies Σ containing both TGDs and EGDs is performed on a set Σ' derived from Σ and containing only TGDs. The aim is to "simulate" the behavior of the EGDs by means of TGDs only. The first approach of this kind, known as *natural simulation*, has been proposed in [21], and further refined by the *substitution-free simulation* in [30]. Below is an example showing how the substitution-free simulation works.

TGDs	TGDs and EGDs
$\mathsf{CT}^{\mathsf{obl}}_{\forall} = \mathsf{CT}^{\mathsf{obl}}_{\exists}$ $\mathsf{CT}^{\mathsf{sobl}}_{\forall} = \mathsf{CT}^{\mathsf{sobl}}_{\exists}$	$\mathsf{CT}^{\mathsf{obl}}_{\forall} \subsetneq \mathsf{CT}^{\mathsf{obl}}_{\exists}$ $\mathsf{CT}^{\mathsf{sobl}}_{\forall} \subsetneq \mathsf{CT}^{\mathsf{sobl}}_{\exists}$
$\mathsf{CT}^{\mathsf{obl}}_{\exists} \subsetneq \mathsf{CT}^{\mathsf{sobl}}_{\forall}$ $\mathsf{CT}^{\mathsf{sobl}}_{\exists} \subsetneq \mathsf{CT}^{\mathsf{std}}_{\forall}$	$\mathsf{CT}^{\mathsf{obl}}_{\exists} \not\parallel \mathsf{CT}^{\mathsf{sobl}}_{\forall}$ $\mathsf{CT}^{\mathsf{sobl}}_{\exists} \not\parallel \mathsf{CT}^{\mathsf{std}}_{\forall}$ $\mathsf{CT}^{\mathsf{obl}}_{\exists} \not\parallel \mathsf{CT}^{\mathsf{std}}_{\forall}$
$\mathsf{CT}^{\mathsf{std}}_{\forall} \subsetneq \mathsf{CT}^{\mathsf{std}}_{\exists}$	$\mathsf{CT}^{\mathsf{std}}_{\forall} \subsetneq \mathsf{CT}^{\mathsf{std}}_{\exists}$
$\mathsf{CT}^{\mathsf{core}}_{\forall} = \mathsf{CT}^{\mathsf{core}}_{\exists}$	$\mathsf{CT}^{\mathsf{core}}_{\forall} = \mathsf{CT}^{\mathsf{core}}_{\exists}$

Table 1: Relationships among the $\mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$'s classes.

EXAMPLE 8. Consider the following set of dependencies Σ_8 (containing both TGDs and EGDs):

$$\begin{array}{lll} r_1: & A(x) \wedge B(x) & \rightarrow \quad C(x) \\ r_2: & C(x) & \rightarrow \quad \exists y\, A(x) \wedge B(y) \\ r_3: & C(x) & \rightarrow \quad \exists y\, A(y) \wedge B(x) \\ r_4: & A(x) \wedge A(y) & \rightarrow \quad x = y \\ r_5: & B(x) \wedge B(y) & \rightarrow \quad x = y \end{array}$$

The substitution-free simulation works as follows:

1. The TGDs below (equality-axioms) are added to Σ_8:

$$\begin{array}{lll} a_1: & Eq(x,y) & \rightarrow \quad Eq(y,x) \\ a_2: & Eq(x,y) \wedge Eq(y,z) & \rightarrow \quad Eq(x,z) \\ a_{3.1}: & A(x) & \rightarrow \quad Eq(x,x) \\ a_{3.2}: & B(x) & \rightarrow \quad Eq(x,x) \\ a_{3.3}: & C(x) & \rightarrow \quad Eq(x,x) \end{array}$$

2. Every occurrence of $x = y$ in Σ_8 is replaced with $Eq(x,y)$. In our case, this affects r_4 and r_5 only, which are replaced with:

$$\begin{array}{lll} r'_4: & A(x) \wedge A(y) & \rightarrow \quad Eq(x,y) \\ r'_5: & B(x) \wedge B(y) & \rightarrow \quad Eq(x,y) \end{array}$$

3. Dependency r_1, which contains multiple occurrences of x in the body, is (non-deterministically) replaced with one of the following two dependencies, where one of the two occurrences of x is replaced with x_2, and the atom $Eq(x,x_2)$ is added to the body:

$$\begin{array}{lll} r'_1: & A(x_2) \wedge B(x) \wedge Eq(x,x_2) & \rightarrow \quad C(x) \\ r''_1: & A(x) \wedge B(x_2) \wedge Eq(x,x_2) & \rightarrow \quad C(x) \end{array}$$

Notice that the only dependencies that remain unchanged are r_2 and r_3. Also, notice that there are no EGDs anymore in the resulting set of dependencies (their role is "simulated" by the rewriting). □

Although not explicitly stated, but somehow left implicit in [21, 30], the natural simulation and the substitution-free simulation ensure the desirable *soundness* property: if, for every database D, all c-chase sequences of D with Σ' are terminating, then for every database D, all c-chase sequences of D with Σ are terminating, for $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$. The natural question now is whether these simulations are also *complete*, that is, if the implication in the opposite direction holds. The answer is negative for both approaches, as stated in the following theorem. Furthermore, we show that the same properties hold when checking for the existence of at least one terminating c-chase sequence. We focus on the substitution-free simulation only, as it is a refinement of the natural simulation.

THEOREM 2. *Let Σ be a set of TGDs and EGDs and Σ' be a set of TGDs obtained from Σ by applying the substitution-free simulation. For every $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$ and every $\mathsf{q} \in \{\forall, \exists\}$,*

1. *if $\Sigma' \in \mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$ then $\Sigma \in \mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$.*
2. *$\Sigma \in \mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$ does not imply $\Sigma' \in \mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$.* □

The theorem above says that there are sets Σ of TGDs and EGDs such that $\Sigma \in \mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$ but their substitution-free simulation Σ' does not belong to $\mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$, and thus it is not possible to realize that $\Sigma \in \mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$ with an analysis of Σ'. The set of dependencies Σ_8 of Example 8 above is one of such cases: Σ_8 belongs to $\mathsf{CT}^{\mathsf{c}}_{\forall}$ (and thus belongs to $\mathsf{CT}^{\mathsf{c}}_{\exists}$ too), but any of its substitution-free simulations is not even in $\mathsf{CT}^{\mathsf{c}}_{\exists}$, for every $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$. The problem is that the simulation of EGDs by means of TGDs is not able to fully capture the specific behavior of EGDs, which replace null values (with constants and other null values). This aspect is not faithfully modeled by storing the information that a null value is equal to a constant or to another null value.

In Sections 5 and 6, we propose approaches that perform a direct analysis of EGDs. However, dealing with EGDs needs some care. In some cases the presence of EGDs allows us to have a terminating c-chase sequence when the set consisting only of the TGDs does not have one; at the same time, the opposite case can occur, that is, in the presence of EGDs there is no terminating c-chase sequence while the set consisting only of the TGDs does have one, where c can be one of $\{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$. The following two examples show such cases.

EXAMPLE 9. Consider the set of dependencies Σ_1 of Example 1 and the database $D = \{N(a)\}$. There is no terminating c-chase sequence of D_1 with the set of TGDs $\Sigma'_1 = \{r_1, r_2\}$, for every $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$. In fact, it is easy to see that an infinite number of facts is introduced: $E(a, \eta_1)$, $N(\eta_1)$, $E(\eta_1, \eta_2), \ldots$. However, the addition of the EGD r_3 allows us to have a terminating c-chase sequence, obtained by enforcing first r_1 and then r_3, and whose result is the universal model $\{N(a), E(a, a)\}$. □

EXAMPLE 10. Consider the set of dependencies Σ_{10} below:

$$r_1 : \; N(x) \rightarrow \exists y \; \exists z \; E(x, y, z)$$
$$r_2 : \; E(x, y, y) \rightarrow N(y)$$
$$r_3 : \; E(x, y, z) \rightarrow y = z$$

For every database D, every c-chase sequence of D with the set of TGDs $\Sigma'_{10} = \{r_1, r_2\}$ is terminating, for every $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$. On the other hand, there is no terminating c-chase sequence of $D = \{N(a)\}$ with Σ_{10}, as an infinite number of facts is introduced: $E(a, \eta_1, \eta_1)$, $N(\eta_1), E(\eta_1, \eta_2, \eta_2)$, $N(\eta_2), \ldots$. □

In the rest of the paper, given a set of dependencies Σ, we use Σ_{tgd} and Σ_{egd} to denote the sets of all TGDs and all EGDs in Σ, respectively (obviously, $\Sigma = \Sigma_{tgd} \cup \Sigma_{egd}$). Furthermore, we use $\Sigma_\forall$ and $\Sigma_\exists$ to denote the set of all full dependencies in Σ (these include full TGDs and all EGDs) and the set of all existentially quantified dependencies in Σ, respectively (obviously, $\Sigma = \Sigma_\forall \cup \Sigma_\exists$).

Recall that for a termination criterion C, we use $\mathcal{C}$ to denote the class of all sets of dependencies recognized by C. For a criterion C defined for TGDs only (e.g., *SwA* and *LS*), we use $\mathcal{C}$ to denote the class of all sets of dependencies Σ such that the set of TGDs obtained from Σ by applying the substitution-free simulation is recognized by C.

5. SEMI-STRATIFICATION

In this section, we introduce a new sufficient condition for checking if a set of dependencies belongs to $\mathsf{CT}^{\mathsf{std}}_{\exists}$. Our condition strictly generalizes stratification.

First of all, we recall the notion of stratification proposed in [13]. Given two dependencies r_1 and r_2, we write $r_1 \prec r_2$ iff there exist an instance K, an instance J, a homomorphism h_1 from $Body(r_1)$ to K, and a homomorphism h_2 from $Body(r_2)$ to J, such that:

- $K \models h_2(r_2)$,
- $K \xrightarrow{r_1, h_1, \gamma_1} J$ is a standard chase step (for some γ_1), and
- $J \not\models h_2(r_2)$.

The *chase graph* $G(\Sigma)$ of a set of dependencies Σ is a directed graph (Σ, E) containing an edge (r_1, r_2) iff $r_1 \prec r_2$. Then, Σ is *stratified* (*Str*) iff every cycle of $G(\Sigma)$ is weakly acyclic.

We now introduce a new relation between dependencies along with the corresponding graph it induces—they are used to define our criterion, allowing us to extend stratification.

DEFINITION 2 (FIRING GRAPH). Let Σ be a set of dependencies. Given two dependencies $r_1, r_2 \in \Sigma$, we write $r_1 < r_2$ iff there exist instances K and J, a homomorphism h_1 from $Body(r_1)$ to K, and a homomorphism h_2 from $Body(r_2)$ to J, such that:

- $K \models h_2(r_2)$,
- $K \xrightarrow{r_1, h_1, \gamma_1} J$ is a standard chase step (for some γ_1),
- $J \not\models h_2(r_2)$, and
- if $r_2 \in \Sigma_\exists$, then $\nexists r_3 \in \Sigma_\forall$ such that $K \xrightarrow{r_3, h_3, \gamma_3} J'$ and $J' \models h_2(r_2)$ (for some h_3, γ_3).

The *firing graph* $G_f(\Sigma)$ of Σ is a directed graph (Σ, E_f) containing a directed edge (r_1, r_2) iff $r_1 < r_2$.

We say that a dependency $r_1 \in \Sigma$ is *fireable with respect to* Σ if there exists a dependency $r_2 \in \Sigma$ such that $r_2 < r_1$. □

DEFINITION 3 (SEMI-STRATIFIED DEPENDENCIES). A set of dependencies Σ is *semi-stratified* (*S-Str*) iff every strongly connected component of $G_f(\Sigma)$ is weakly acyclic. □

EXAMPLE 11. Consider the following set of TGDs Σ_{11}:

$$r_1 : \; N(x) \rightarrow \exists y \; E(x, y)$$
$$r_2 : \; E(x, y) \rightarrow N(y)$$
$$r_3 : \; E(x, y) \rightarrow E(y, x)$$

The chase and the firing graphs are depicted in Figure 1. Notice that, since r_2 and r_3 are full TGDs, their incoming edges are the same in the two graphs. On the other hand, the edge in $G(\Sigma_{11})$ from r_2 to r_1 does not belong to $G_f(\Sigma_{11})$, as the firing of r_1 because of r_2 is blocked by first enforcing r_3. It can be easily verified that Σ_{11} is semi-stratified, but not stratified.

Consider now the database $D = \{N(a)\}$. The standard chase sequence consisting of the iterative application of r_1 followed by r_2 is non-terminating. However, if we apply

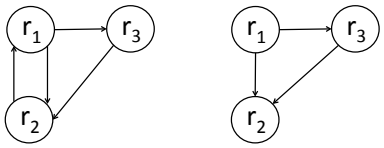

Figure 1: Chase graph (left) and firing graph (right) of Σ_{11}.

r_3 before r_1, we obtain a terminating standard chase sequence producing the instance $K = \{N(a), E(a, \eta_1), N(\eta_1), E(\eta_1, a)\}$. Such a standard chase sequence is terminating as no more standard chase steps can be added. □

THEOREM 3. *For every semi-stratified set of dependencies Σ and for every database D, there exists a terminating standard chase sequence of D with Σ whose length is polynomial in the size of D.* □

As the following theorem states, it can be decided in coNP whether a set of dependencies is semi-stratified.

THEOREM 4. *Deciding if a set of dependencies is semi-stratified is in coNP.* □

The following theorem shows the relative expressivity of $\mathcal{S}$-$\mathcal{S}tr$ and other classes of dependencies previously proposed.

THEOREM 5.

1. $\mathcal{S}tr \subsetneq \mathcal{S}$-$\mathcal{S}tr$.
2. $\mathcal{S}$-$\mathcal{S}tr \not\parallel \mathcal{C}$ *for* $\mathcal{C} \in \{\mathcal{SC}, \mathcal{AC}, \mathcal{MFA}\}$.

Notice that SC, AC, and MFA guarantee that all standard chase sequences are terminating, while Str and S-Str guarantee the existence of at least one terminating standard chase sequence.

We recall that $\mathcal{SC} \subsetneq \mathcal{AC}$, and thus the incomparability of $\mathcal{S}$-$\mathcal{S}tr$ with $\mathcal{SC}$ and $\mathcal{AC}$ implies that $\mathcal{S}$-$\mathcal{S}tr$ is incomparable also with any other class included by $\mathcal{AC}$ and containing $\mathcal{SC}$ (e.g., $\mathcal{SwA}$, $\mathcal{SR}$, and $\mathcal{IR}$)—see [27] for a complete picture.

6. ADORNMENT ALGORITHM

In this section, we propose another decidable sufficient condition for a set of dependencies to be in $\mathsf{CT}^{\mathsf{std}}_{\exists}$.

Specifically, we propose an algorithm which takes as input a set of dependencies, and gives as output a set of adorned dependencies and a boolean value. The aim of the algorithm is twofold: *(i)* it defines a termination criterion on its own—on the basis of the boolean value returned by the algorithm; and *(ii)* it can be combined with other termination criteria to enhance them, in that (strictly) more sets of dependencies in $\mathsf{CT}^{\mathsf{std}}_{\exists}$ can be identified by using our algorithm in conjunction with a termination criterion—this is achieved by analyzing the set of adorned dependencies returned by the algorithm. Before presenting our approach, we introduce additional terminology and notation.

Adornments. An *adornment symbol* is an element of the alphabet $\Lambda = \{b\} \cup \{f_i \mid i \in \mathbb{N}\}$, where b is called "bound" symbol and the f_i's are called "free" symbols. Consider an n-ary predicate R. An *adornment* of R is a string α of length n built from adornment symbols; we call R^α an *adorned predicate*. An *adorned atom* is of the form $R^\alpha(\mathbf{t})$, where $R(\mathbf{t})$ is an atom and α is an adornment of R. An *adorned conjunction* is a conjunction of adorned atoms. An *adorned dependency* is a dependency containing adorned atoms. Given an adorned formula (i.e., atom, conjunction of atoms, dependency, etc.) or set of adorned formulas F, we use $src(F)$ to denote the formula or set of formulas derived from F by deleting all adornments. We also say that F is an *adorned version* of $src(F)$.

Given a set of adorned predicates AP, the set of the *adorned versions* of an atom $R(\mathbf{t})$ *w.r.t.* AP is defined as follows:

$$\mathcal{A}(R(\mathbf{t}), AP) = \{R^\alpha(\mathbf{t}) \mid R^\alpha \in AP\}$$

The set of the *adorned versions* of a conjunction of atoms $\varphi = A_1 \wedge \cdots \wedge A_k$ *w.r.t.* AP is defined as follows:

$$\mathcal{A}(\varphi, AP) = \{A_1^{\alpha_1} \wedge \cdots \wedge A_k^{\alpha_k} \mid A_i^{\alpha_i} \in \mathcal{A}(A_i, AP) \text{ for } 1 \leq i \leq k\}$$

If φ is the empty conjunction, then $\mathcal{A}(\varphi, AP)$ contains only the empty conjunction.

Given an adorned atom $R^{\alpha_1 \cdots \alpha_n}(t_1, \ldots, t_n)$, we say that t_i *is adorned with* α_i. An adorned atom or conjunction is *coherent* if every variable occurring in it is always adorned with the same adornment symbol and constants are adorned with b. For instance, the adorned conjunction $N^b(x) \wedge E^{f_1 b}(x, y)$ is not coherent because x is adorned with b in the first atom and with f_1 in the second atom. On the other hand, $N^{f_1}(x) \wedge E^{f_1 b}(x, y)$ is coherent.

An *adornment definition* is an expression of the form $f_i = f^r_z(\alpha)$ where f_i is an adornment symbol, r is a TGD of the form $\varphi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x}, \mathbf{z})$, z is in $\mathbf{z}$, and α is a string of n adornment symbols with n being the number of variables in $\mathbf{x}$. The role of adornment definitions will be explained shortly.

Head adornment. One important step of our adornment algorithm is the propagation of adornments from the body to the head of dependencies, which is defined as follows. Given a set AD of adornment definitions, a dependency $r : body \rightarrow head$, and a coherent adorned version $body^\mu$ of $body$, we define $HeadAdn(r, body^\mu, AD)$ as the procedure that updates AD and returns an adorned version $head^\mu$ of $head$ as follows:

1. if r is an EGD, then $head^\mu = head$, and AD is not modified.
2. Otherwise, r is a TGD $\varphi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x}, \mathbf{z})$ and $head^\mu$ is obtained from $\exists \mathbf{z}\, \psi(\mathbf{x}, \mathbf{z})$ as follows:
   - every universally quantified variable (i.e., every $x \in \mathbf{x}$) is adorned with the same adornment symbol the variable is adorned with in $body^\mu$ (notice that such an adornment symbol is unique as $body^\mu$ is coherent);
   - constants are adorned with b;
   - every (existentially quantified) variable $z \in \mathbf{z}$ is adorned as follows.[1] Let $f^r_z(\alpha)$ be the Skolem term where if $\mathbf{x} = x_1, \ldots, x_n$ then $\alpha = \alpha_1, \ldots, \alpha_n$ is the string of adornment symbols such that every x_j is adorned with α_j in $body^\mu$, for $1 \leq j \leq n$. If an adornment definition of the form $f_i = f^r_z(\alpha)$ is already in AD, then z is adorned with f_i and AD is not modified. Otherwise, z is adorned with f_j, where $j = 1 + \max\{k \mid f_k \text{ appears in } AD\}$, and $f_j = f^r_z(\alpha)$ is added to AD.

[1]It is assumed that the existentially quantified variables are considered one at a time following the order they appear in $\mathbf{z}$. Also, an arbitrary but fixed ordering of the variables in $\mathbf{x}$ is assumed.

For instance, assuming $AD = \emptyset$ and given a TGD $r : R(x,y) \rightarrow \exists z\ R(x,z)$, then $HeadAdn(r, R^{bb}(x,y), AD)$ gives the adorned formula $\exists z\, R^{bf_1}(x,z)$ and $f_1 = f_z^r(b)$ is added to AD.

Cyclic adornment symbol. Given a set of adornment definitions AD, we use $\Omega(AD)$ to denote the labeled directed graph whose vertices are the adornment symbols appearing in AD, and where there is a directed edge from f_i to f_j labeled with f_z^r iff there are $f_i = f_z^r(\cdots f_j \cdots)$ and $f_j = f_w^s(\cdots)$ in AD with $r, s \in \Sigma_\exists$ and there are $r_1, ..., r_n \in \Sigma_\forall$ $(n \geq 0)$ such that $s < r_1 < \cdots < r_n < r$.

An adornment symbol f_i is *cyclic w.r.t. AD* if there is a path in $\Omega(AD)$ departing from f_i where (at least) two edges have the same label. We say that an adorned head $\exists \mathbf{z}\, \psi^\mu(\mathbf{x}, \mathbf{z})$ is *cyclic* (*w.r.t. AD*) if there is a variable z in $\mathbf{z}$ adorned with a cyclic adornment symbol.

Adornment Substitution. An *adornment substitution* θ is a set of pairs of the form f_i/f_j (whose intuitive meaning is that f_i is replaced by f_j), where f_i and f_j are adornment symbols such that if $f_i/f_j \in \theta$ then there is no f_j/f_k in θ (that is, a symbol f_j used to replace a symbol f_i cannot be substituted by a symbol f_k). The result of applying θ to an adornment α, denoted $\alpha\theta$, is the adornment obtained from α by simultaneously replacing every occurrence in α of an adornment symbol f_i with f_j iff $f_i/f_j \in \theta$. This is extended to adorned atoms, adorned dependencies, adornment definitions, etc., in the obvious way.

Given a set of adornment definitions AD, an adornment substitution θ is *valid* (*w.r.t. AD*) if for every f_i/f_j in θ, it is the case that AD contains adornment definitions of the form $f_i = f_z^r(\alpha)$ and $f_j = f_z^r(\alpha')$.

Given a set of adorned dependencies Σ^μ and a dependency r, we define:

$$AP(\Sigma^\mu) = \{R^\alpha \mid R^\alpha(\mathbf{t}) \text{ appears in } \Sigma^\mu\}$$
$$D^\mu(\Sigma^\mu) = \{R(\alpha_1, ..., \alpha_n) \mid R^{\alpha_1 \cdots \alpha_n} \in AP(\Sigma^\mu)\}$$
$$B^\mu(r, \Sigma^\mu) = \{body^\mu \mid r^\mu : body^\mu \rightarrow head^\mu \in \Sigma^\mu \wedge src(r^\mu) = r\}$$

We are now ready to introduce the $Adn^\exists$ algorithm (Algorithm 1). The input is a set of dependencies Σ, while the output is a set of adorned dependencies Σ^μ along with a boolean value *Acyc*. As mentioned before, the aim of the algorithm is twofold: it defines a termination criterion on its own, and it can be combined with other termination criteria.

More specifically, if *Acyc* is *false*, then a form of cyclicity has been detected; otherwise, for every database D, there is a terminating standard chase sequence of D with Σ.

As for the second aim of the algorithm, the adorned set of dependencies Σ^μ given as output can be used as follows: a sufficient condition for checking membership in $\mathsf{CT}^{\mathsf{std}}_\exists$ is applied to Σ^μ rather than Σ. If Σ^μ satisfies the condition, then the original set of dependencies Σ is in $\mathsf{CT}^{\mathsf{std}}_\exists$.

The basic idea of the algorithm is to produce adorned dependencies from the original ones by keeping track of what facts can be derived by a chase execution and how terms are derived. When adorning dependencies, the algorithm's strategy is to adorn first full dependencies, and to adorn existentially quantified dependencies only when no further full dependency can be adorned. This is iterated as long as new adorned dependencies can be derived. EGDs are leveraged to see if free symbols can be changed.

The algorithm maintains two sets Σ^μ and AD, containing the adorned dependencies and the adornment definitions currently derived, respectively. These two sets are also used by Function 2, which is called by Algorithm 1 to verify whether a dependency r can be adorned, on the basis of Σ^μ and AD (these are not explicitly passed to Function 2, but are treated as "global variables"). Specifically, to see if a dependency $r = body \rightarrow head$ can be adorned, function *adorn* proceeds as follows. It checks if there is a coherent adorned version $body^\mu$ of $body$ (obtained using adorned predicates in $AP(\Sigma^\mu)$) such that there is no dependency in Σ^μ having $body^\mu$ as body. If such a coherent adorned version $body^\mu$ exists, the adorned head $head^\mu = HeadAdn(r, body^\mu, AD(\Sigma^\mu))$ is computed, by propagating adornments from $body^\mu$. If $r^\mu = body^\mu \rightarrow head^\mu$ is fireable w.r.t. Σ^μ, then r^μ can be added to Σ^μ, and thus is returned along with the boolean value *true*. Otherwise, the input dependency r is returned along with the boolean value *false*.

We now go into the details of Algorithm 1. Initially, *Acyc* is *true*, AD is empty, and Σ^μ contains a dependency $R(x_1, ..., x_n) \rightarrow R^{b \cdots b}(x_1, ..., x_n)$ for each $R \in \mathbf{R}$ (lines 1–3). As the algorithm proceeds, Σ^μ and AD are extended and modified; in the case a form of cyclicity is detected the value of *Acyc* is changed to *false*. Specifically, the algorithm proceeds as follows (until Σ^μ does not change).

It first checks if there is a universally quantified dependency r that can be adorned (line 6), using function *adorn*. If this is the case, the corresponding adorned dependency r^μ is added to Σ^μ (line 7). Moreover, if r is an EGD and is not satisfied by $D^\mu(\Sigma^\mu)$, then the *ChaseStep* function executes a chase step over $D^\mu(\Sigma^\mu)$ with r (line 9). Notice that facts in $D^\mu(\Sigma^\mu)$ contains bound (i.e., b's) and free (i.e., f_i's) symbols: the former is treated as a constant while the latter are treated as labeled nulls. If the chase step replaces f_i with s, where s is either b or an f_j with $i \neq j$, then $\tau = \{f_i/s\}$.[2] Finally, τ is applied to Σ^μ, all the definitions of f_i are deleted from AD, and τ is applied to AD—to replace occurrences of f_i in the right-hand side of adornment definitions (line 10).

When there are no full dependencies that can be adorned, the algorithm checks if there is an existentially quantified dependency that can be adorned (line 11), and if so, a corresponding adorned dependency r^μ is added to Σ^μ (line 12).

After a dependency is adorned into r^μ, the algorithm checks if there exists a non-empty valid substitution θ s.t. $r^\mu\theta$ is equal to r^υ for some r^υ in Σ^μ (line 13). If this is the case, then θ is applied to Σ^μ and AD (line 14). This ensures termination of $Adn^\exists$. Moreover, if $head^\mu\theta$ is cyclic, then a form of cyclicity that may lead to non-termination is detected and *Acyc* is set to *false* (line 16).

The overall process described so far is iterated as long as Σ^μ changes.

Example 12. Consider the set of dependencies Σ_1 of Example 1. Initially, the following two adorned dependencies, mapping unadorned atoms to atoms adorned with strings of b's, are added to Σ_1^μ:

$$s_1 : \quad N(x) \rightarrow N^b(x)$$
$$s_2 : \quad E(x,y) \rightarrow E^{bb}(x,y)$$

The algorithm then proceeds by adorning full dependencies and adds the following adorned dependencies to Σ_1^μ:

$$s_3 : \quad E^{bb}(x,y) \rightarrow x = y$$
$$s_4 : \quad E^{bb}(x,y) \rightarrow N^b(x)$$

[2]With a slight abuse of notation, here we allow adornment substitutions containing f_i/b.

Algorithm 1 $Adn^{\exists}$

Input: Set of dependencies Σ over schema $\mathbf{R}$.
Output: Set of adorned dependencies Σ^{μ}, Boolean value $Acyc$.

$Acyc = true$;
$\Sigma^{\mu} = \{R(x_1, ..., x_n) \rightarrow R^{b...b}(x_1, ..., x_n) \mid R \in \mathbf{R} \text{ and } ar(R) = n\}$;
$AD = \emptyset$;
repeat
  $\Sigma^{\mu}_{old} = \Sigma^{\mu}$;
  if $\exists r \in \Sigma_{\forall}$ s.t. $\langle b, r^{\mu} \rangle = adorn(r)$ and $b = true$ **then**
    $\Sigma^{\mu} = \Sigma^{\mu} \cup \{r^{\mu}\}$;
    if $r \in \Sigma_{egd}$ s.t. $D^{\mu}(\Sigma^{\mu}) \not\models r$ **then**
      $\tau = \{f_i/s\} = ChaseStep(r, D^{\mu}(\Sigma^{\mu}))$;
      $\Sigma^{\mu} = \Sigma^{\mu}\tau$; $AD = AD \backslash \{f_i = f^{s}_{z}(\alpha) \in AD\}$; $AD = AD\tau$;
  else if $\exists r \in \Sigma_{\exists}$ s.t. $\langle b, r^{\mu} \rangle = adorn(r)$ and $b = true$ **then**
    $\Sigma^{\mu} = \Sigma^{\mu} \cup \{r^{\mu}\}$;
  if $\exists\, r^{\upsilon} \in \Sigma^{\mu} \wedge \exists$valid subst. $\theta \neq \emptyset$ s.t. $r^{\mu}\theta = r^{\upsilon} \wedge src(r^{\upsilon}) = r$ **then**
    $\Sigma^{\mu} = \Sigma^{\mu}\theta$; $AD = AD\theta$;
    if $head^{\mu}\theta$ is cyclic **then**
      $Acyc = false$;
until $\Sigma^{\mu} = \Sigma^{\mu}_{old}$
return $\langle \Sigma^{\mu}, Acyc \rangle$;

Function 2 *adorn*

Input: Dependency $r = body \rightarrow head$.
Output: Pair $\langle bool, r' \rangle$, where $bool$ is a Boolean value and r' is a possibly adorned dependency.

if $\exists body^{\mu} \in \mathcal{A}(body, AP(\Sigma^{\mu}))$ s.t.
  a) $body^{\mu}$ is coherent,
  b) $body^{\mu} \notin B^{\mu}(r, \Sigma^{\mu})$, and
  c) $r^{\mu} = body^{\mu} \rightarrow head^{\mu}$ is fireable w.r.t. Σ^{μ},
  where $head^{\mu} = HeadAdn(r, body^{\mu}, AD)$ **then**
  return $\langle true, r^{\mu} \rangle$;
else
  return $\langle false, r \rangle$;

Notice that $D^{\mu}(\Sigma^{\mu}_1) = \{N(b), E(b, b)\}$ and thus the EGD r_3 in Σ_1 is satisfied by $D^{\mu}(\Sigma^{\mu}_1)$. Next, the existentially quantified dependency (namely, r_1) is adorned and the following adorned dependency is added to Σ^{μ}_1:

$$s_5: \quad N^b(x) \quad \rightarrow \quad \exists y\ E^{bf_1}(x, y)$$

Moreover, $AD = \{f_1 = f^{r_1}_{y}(b)\}$. After that, the algorithm starts considering full dependencies again. By adorning the EGD r_3, the following adorned dependency is obtained, which is added to Σ^{μ}_1:

$$s_6: \quad E^{bf_1}(x, y) \quad \rightarrow \quad x = y$$

Notice that $D^{\mu}(\Sigma^{\mu}_1) = \{N(b), E(b, b), E(b, f_1)\}$ and thus $D^{\mu}(\Sigma^{\mu}_1) \not\models r_3$. Thus, function *ChaseStep* is executed with $D^{\mu}(\Sigma^{\mu}_1)$ and r_3, returning the substitution $\theta = \{f_1/b\}$, which is applied to Σ^{μ}_1, whereas AD becomes empty. After the application of θ, we have that $\Sigma^{\mu}_1 = \{s_1, s_2, s_3, s_4, s'_5\}$, where s'_5 is derived from s_5 by replacing f_1 with b, that is, $s'_5 : N^b(x) \rightarrow \exists y\ E^{bb}(x, y)$.

At this point, no further dependencies can be adorned and the algorithm terminates by returning the value $Acyc = true$ along with Σ^{μ}_1. Notice that there is no dependency (of any kind) that can be adorned because $AP(\Sigma^{\mu}_1) = \{N^b, E^{bb}\}$ and the body of the dependencies in Σ_1 have already been adorned using these adorned predicates. □

Example 13. Consider the set of dependencies Σ_{10} of Example 10. Initially, the following adorned dependencies are added to Σ^{μ}_{10}:

$$s_1: N(x) \rightarrow N^b(x)$$
$$s_2: E(x, y, z) \rightarrow E^{bbb}(x, y, z)$$

Then, full dependencies are adorned and the following adorned dependencies are added to Σ^{μ}_{10}:

$$s_3: E^{bbb}(x, y, z) \rightarrow y = z$$
$$s_4: E^{bbb}(x, y, y) \rightarrow N^b(y)$$

Notice that $D^{\mu}(\Sigma^{\mu}_{10}) = \{N(b), E(b, b, b)\}$ and thus the EGD r_3 in Σ_{10} is satisfied by $D^{\mu}(\Sigma^{\mu}_{10})$. Next, the existentially quantified dependency (namely, r_1) is adorned and the following adorned dependency is added to Σ^{μ}_{10}:

$$s_5: N^b(x) \rightarrow \exists y\ \exists z\ E^{bf_1f_2}(x, y, z)$$

with $AD = \{f_1 = f^{r_1}_{y}(b), f_2 = f^{r_1}_{z}(b)\}$. Now universally quantified dependencies are considered again to see if they can be adorned. Suppose r_3 is chosen. Then, the following adorned dependency is added to Σ^{μ}_{10}:

$$s_6: E^{bf_1f_2}(x, y, z) \rightarrow y = z$$

Now, $D^{\mu}(\Sigma^{\mu}_{10}) = \{N(b), E(b, b, b), E(b, f_1, f_2)\}$, which does not satisfy the EGD r_3. By executing the *ChaseStep* function on $D^{\mu}(\Sigma^{\mu}_{10})$ and r_3, the substitution $\tau = \{f_2/f_1\}$ is obtained (alternatively, f_1/f_2 might have been chosen, but the choice is immaterial). Then, the adornment definition $f_2 = f^{r_1}_{z}(b)$ is removed from AD, and the substitution τ is applied to both Σ^{μ}_{10} and AD, replacing f_2 with f_1. Thus, AD becomes $\{f_1 = f^{r_1}_{y}(b)\}$, while s_5 and s_6 become:

$$s'_5: N^b(x) \rightarrow \exists y\ \exists z\ E^{bf_1f_1}(x, y, z)$$
$$s'_6: E^{bf_1f_1}(x, y, z) \rightarrow y = z$$

By proceeding as discussed above, the following adorned dependencies are added to Σ^{μ}_{10}:

$$s_7: E^{bf_1f_1}(x, y, y) \rightarrow N^{f_1}(y)$$
$$s_8: N^{f_1}(x) \rightarrow \exists y\ \exists z\ E^{f_1f_3f_3}(x, y, z)$$
$$s_9: E^{f_1f_3f_3}(x, y, z) \rightarrow y = z$$
$$s_{10}: E^{f_1f_3f_3}(x, y, y) \rightarrow N^{f_3}(y)$$
$$s_{11}: N^{f_3}(x) \rightarrow \exists y\ \exists z\ E^{f_3f_5f_5}(x, y, z)$$
$$s_{12}: E^{f_3f_5f_5}(x, y, z) \rightarrow y = z$$
$$s_{13}: E^{f_3f_5f_5}(x, y, y) \rightarrow N^{f_5}(y)$$
$$s_{14}: N^{f_5}(x) \rightarrow \exists y\ \exists z\ E^{f_5f_7f_7}(x, y, z)$$
$$s_{15}: E^{f_5f_7f_7}(x, y, z) \rightarrow y = z$$

with $AD = \{f_1 = f^{r_1}_{y}(b),\ f_3 = f^{r_1}_{y}(f_1),\ f_5 = f^{r_1}_{y}(f_3),\ f_7 = f^{r_1}_{y}(f_5)\}$. When s_{15} is introduced, a valid substitution $\theta = \{f_5/f_1, f_7/f_3\}$ mapping s_{15} to s_9 is found. Thus, θ is applied to both Σ^{μ}_{10} and AD, replacing all occurrences of adornment symbols f_5 and f_7 with f_1 and f_3, respectively. Notice that dependencies $s_{11} - s_{14}$ become:

$$s'_{11}: N^{f_3}(x) \rightarrow \exists y\ \exists z\ E^{f_3f_1f_1}(x, y, z)$$
$$s'_{12}: E^{f_3f_1f_1}(x, y, z) \rightarrow y = z$$
$$s'_{13}: E^{f_3f_1f_1}(x, y, y) \rightarrow N^{f_1}(y)$$
$$s'_{14}: N^{f_1}(x) \rightarrow \exists y\ \exists z\ E^{f_1f_3f_3}(x, y, z)$$

while s_{15} becomes equal to s_9. Moreover, $AD = \{f_1 = f^{r_1}_{y}(b),\ f_3 = f^{r_1}_{y}(f_1),\ f_1 = f^{r_1}_{y}(f_3)\}$. Since $\Omega(\Sigma^{\mu}_{10})$ is cyclic (after the application of θ), as it contains the edges (f_1, f_3) and (f_3, f_1), variable $Acyc$ is set to *false*.

At this point, no further dependencies can be adorned and the algorithm terminates by returning the value $Acyc = false$ along with Σ^{μ}_{10}. □

Theorem 6. *Algorithm* $Adn^{\exists}$ *terminates for every set of dependencies.* □

$\lvert\Sigma_{\exists}\rvert$ \ $\lvert\Sigma_{egd}\rvert$	[1, 10]		[11, 100]	
	#tests	$\lvert\Sigma\rvert$	#tests	$\lvert\Sigma\rvert$
[1, 10]	50	86	7	451
[11, 100]	15	406	26	1,210
[101, 1000]	51	3,113	13	3,176
[1001, 5000]	9	9,117	7	19,587

(a) Ontologies' Size

[1, 10]		[11, 100]	
$\lvert\Sigma^{\mu}\rvert/\lvert\Sigma\rvert$	Time	$\lvert\Sigma^{\mu}\rvert/\lvert\Sigma\rvert$	Time
2.38	84	3.15	125
2.45	141	2.83	275
2.97	787	6.16	22,819
2.82	712	2.82	1,495

(b) Complexity

[1, 10]		[11, 100]	
A+NT	FN	A+NT	FN
$\mathbf{50}_{[44+6]}$	**0**	$\mathbf{7}_{[6+1]}$	**0**
$\mathbf{15}_{[6+9]}$	**0**	$\mathbf{26}_{[13+13]}$	**0**
$\mathbf{51}_{[4+47]}$	**0**	$\mathbf{11}_{[1+10]}$	**2**
$\mathbf{9}_{[0+9]}$	**0**	$\mathbf{7}_{[0+7]}$	**0**

(c) Expressivity

Table 2: Experimental Results.

Thus, given an input set of dependencies Σ, Algorithm 1 always returns a pair consisting of a set Σ^{μ} of adorned dependencies and a boolean value *Acyc* giving information about the detection of a form of cyclicity—we use $Adn^{\exists}(\Sigma)[1]$ to refer to Σ^{μ} and $Adn^{\exists}(\Sigma)[2]$ to refer to *Acyc*.

Another important property of Algorithm 1 is stated in the next theorem. It says that, given a set of dependencies Σ and a database D, some of the canonical models of (D, Σ) can be obtained from the canonical models of (D, Σ^{μ}) by dropping adornments, where $\Sigma^{\mu} = Adn^{\exists}(\Sigma)[1]$. Moreover, whenever (D, Σ) has canonical models, (D, Σ^{μ}) admits canonical models as well. These two properties imply that if (D, Σ) has canonical models, then we can construct one from (D, Σ^{μ}) (e.g., by using the core chase).

THEOREM 7. *Consider a set of dependencies Σ and let $\Sigma^{\mu} = Adn^{\exists}(\Sigma)[1]$. For every database D,*

1. *$src(CMod(D, \Sigma^{\mu})) \subseteq CMod(D, \Sigma)$, and*
2. *$CMod(D, \Sigma^{\mu}) \neq \emptyset$ iff $CMod(D, \Sigma) \neq \emptyset$.* □

On the basis of the boolean value returned by Algorithm 1, below we define *semi-acyclic* dependencies.

DEFINITION 4 (SEMI-ACYCLIC DEPENDENCIES). A set of dependencies Σ is *semi-acyclic (SAC)* if $Adn^{\exists}(\Sigma)[2]$ is true. □

Every semi-acyclic set of dependencies belongs to $\mathsf{CT}^{\mathsf{std}}_{\exists}$.

THEOREM 8. *For every semi-acyclic set of dependencies Σ and for every database D, there is a terminating standard chase sequence of D with Σ whose length is polynomial in the size of D.* □

7. EXPRESSIVITY, COMPLEXITY, AND EXPERIMENTAL EVALUATION

As Algorithm 1 embeds the fireable condition of semi-stratification, we have that semi-acyclicity strictly generalizes semi-stratification. It also generalizes acyclicity.

THEOREM 9. *$\mathcal{S}$-Str $\subsetneq$ $\mathcal{SAC}$ and $\mathcal{AC} \subsetneq \mathcal{SAC}$.* □

As $\mathcal{SAC}$ includes sets of dependencies which are not in $\mathsf{CT}^{\mathsf{std}}_{\forall}$, it follows that $\mathcal{SAC} \not\subseteq \mathcal{MFA}$; it is an open problem whether $\mathcal{MFA} \subseteq \mathcal{SAC}$.

We now turn our attention to the second aim of Algorithm 1: providing a set of adorned dependencies Σ^{μ} which can be used in place of the original set of dependencies Σ for termination analysis. As shown in the following, Σ^{μ} turns out to be better than Σ for the purpose of checking termination (see Theorem 11 below).

Given a termination criterion C, we use $Adn^{\exists}$-$\mathcal{C}$ to denote the class of sets of dependencies Σ such that $Adn^{\exists}(\Sigma)[1]$ belongs to $\mathcal{C}$. Moreover, we define $\mathbf{C}$ as the set containing $\mathcal{C}$ for every criterion C discussed in Section 3.

The following theorem states that by combining Algorithm 1 with current termination criteria (including those for checking if a set of dependencies belongs to $\mathsf{CT}^{\mathsf{std}}_{\forall}$), we can check (via a sufficient condition) if a set of dependencies belongs to $\mathsf{CT}^{\mathsf{std}}_{\exists}$. Theorem 11 below says that by proceeding in this way we can identify strictly more sets of dependencies in $\mathsf{CT}^{\mathsf{std}}_{\exists}$.

THEOREM 10. *Let Σ be a set of dependencies. If $\Sigma \in Adn^{\exists}$-$\mathcal{C}$ then $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\exists}$, for $\mathcal{C} \in \mathbf{C}$.* □

THEOREM 11. *$\mathcal{C} \subsetneq Adn^{\exists}$-$\mathcal{C}$, for $\mathcal{C} \in \mathbf{C}$.* □

The previous theorem follows from the fact that if a set of dependencies satisfies a termination condition, then its adorned version has the same (or weaker) structural properties and thus it satisfies the termination condition too.

We point out that if $\Sigma \in Adn^{\exists}$-$\mathcal{C}$ then $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\exists}$, but it can be the case that $\Sigma \notin \mathsf{CT}^{\mathsf{std}}_{\forall}$ even if C is a criterion for checking if a set of dependencies is in $\mathsf{CT}^{\mathsf{std}}_{\forall}$.

The following theorem states the complexity of Algorithm 1.

THEOREM 12. *For any set of dependencies Σ, the size of $Adn^{\exists}(\Sigma)[1]$ and the time complexity of computing it using Algorithm 1 are exponential and double exponential in the size of Σ, respectively.*

Despite of the theorem above, as shown in our experimental evaluation, the size of Σ^{μ} and the time to compute it are reasonable in practice.

Experimental Evaluation. We now report on an experimental evaluation we performed to assess our approach. We have implemented Algorithm 1 in Java. The implementation, as well as the datasets we used, can be found at `http://si.deis.unical.it/~calautti/chase/`. We used sets of dependencies taken from the repository [1], which includes ontologies in a variety of domains: a large subset of the Gardiner ontology corpus [18], the LUBM ontology [28], several Phenoscape ontologies [3], and a number of ontologies from two versions of the Open Biomedical Ontology corpus [2]. All experiments were run on an Intel i7-3770 3.40 Ghz, 16 GB of memory.

Table 2 resumes (*a*) the main characteristics of the dependency sets used in our experiments, (*b*) the complexity of analyzing a set of dependencies in terms of the number of generated adorned rules and the time to compute them, and (*c*) the expressive power in terms of the number of sets of dependencies recognized as terminating or not.

More specifically, we considered a collection of 178 ontologies and partitioned it into eight classes depending on

the number of existentially quantified TGDs and the number of EGDs. For the former we considered four intervals, namely $[1, 10]$, $[11, 100]$, $[101, 1000]$ and $[1001, 5000]$, while for the latter we considered two intervals, namely $[1, 10]$ and $[11, 100]$. For each class, we have considered ontologies with different ratios $|\Sigma_\forall|/|\Sigma_\exists|$.

Table 2a reports, for each class, the number of ontologies belonging to the class (column #tests) along with the average number of dependencies for the ontologies in the class (column $|\Sigma|$).

Table 2b shows, for each class, the average ratio of the number of adorned dependencies to the number of dependencies in the original ontology (column $|\Sigma^\mu|/|\Sigma|$), along with the average time (in milliseconds) to compute the adorned set (column Time). It is worth noting that the set of adorned dependencies in not much larger than the original set of dependencies, and running times are lower than 1 second in most of the cases.

Table 2c reports, for each class, *(i)* the number of semi-acyclic ontologies + the number of ontologies that are not semi-acyclic and the standard chase did not halt within 24 hours (column $A + NT$), and *(ii)* the number of ontologies that are not semi-acyclic and the standard chase terminated within 24 hours (column FN, "false negatives"). Notice that, among the 76 ontologies for which the chase terminated, only 2 were not semi-acyclic.

8. REFERENCES

[1] Information Systems Group Ontologies, `http://www.cs.ox.ac.uk/isg/ontologies/`.

[2] The OBO Foundry, `http://www.obofoundry.org`.

[3] Phenoscape Ontologies, `http://phenoscape.org/wiki/Ontologies`.

[4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[5] M. Arenas, P. Barceló, R. Fagin, and L. Libkin. Locally consistent transformations and query answering in data exchange. In *PODS*, pages 229–240, 2004.

[6] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[7] C. Beeri and M. Y. Vardi. Formal systems for tuple and equality generating dependencies. *SIAM J. Comput.*, 13(1):76–98, 1984.

[8] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*, pages 268–279, 2011.

[9] M. Calautti, G. Gottlob, and A. Pieris. Chase termination for guarded existential rules. In *PODS*, pages 91–103, 2015.

[10] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *JAIR*, 48:115–174, 2013.

[11] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.

[12] G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. On reconciling data exchange, data integration, and peer data management. In *PODS*, pages 133–142, 2007.

[13] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

[14] R. Fagin. Equality-generating dependencies. In *Encyclopedia of Database Systems*, pages 1009–1010. 2009.

[15] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Th. Comp. Sc.*, 336(1):89–124, 2005.

[16] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM TODS*, 30(1):174–210, 2005.

[17] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM TODS*, 33(2), 2008.

[18] T. Gardiner, D. Tsarkov, and I. Horrocks. Framework for an automated comparison of description logic reasoners. In *ISWC*, pages 654–667, 2006.

[19] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 6(9):625–636, 2013.

[20] T. Gogacz and J. Marcinkowski. All-instances termination of chase is undecidable. In *ICALP*, pages 293–304, 2014.

[21] G. Gottlob and A. Nash. Efficient core computation in data exchange. *J. ACM*, 55(2), 2008.

[22] G. Grahne and A. Onet. Anatomy of the chase. *CoRR*, abs/1303.6682, 2013.

[23] B. C. Grau, I. Horrocks, M. Krotzsch, C. Kupke, D. Magka, B. Motik, and Z. Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *JAIR*, 47:741–808, 2013.

[24] S. Greco, C. Molinaro, and F. Spezzano. *Incomplete Data and Data Dependencies in Relational Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[25] S. Greco and F. Spezzano. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.

[26] S. Greco, F. Spezzano, and I. Trubitsyna. Stratification criteria and rewriting techniques for checking chase termination. *PVLDB*, 4(11):1158–1168, 2011.

[27] S. Greco, F. Spezzano, and I. Trubitsyna. Checking chase termination: Cyclicity analysis and rewriting techniques. *IEEE TKDE*, 27(3):621–635, 2015.

[28] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[29] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM TODS*, 4(4):455–469, 1979.

[30] B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.

[31] M. Meier. *On the Termination of the Chase Algorithm*. Albert-Ludwigs-Universitat Freiburg, 2010.

[32] M. Meier, M. Schmidt, and G. Lausen. On chase termination beyond stratification. *PVLDB*, 2(1):970–981, 2009.

[33] A. Onet. The chase procedure and its applications in data exchange. In *Data Exchange, Integration, and Streams*, pages 1–37. 2013.