# The Chase Procedure and its Applications in Data Exchange

Adrian Onet

**Concordia University**
**Montreal, Canada**
adrian_onet@yahoo.com

**Abstract**

The initial and basic role of the chase procedure was to test logical implication between sets of dependencies in order to determine equivalence of database instances known to satisfy a given set of dependencies and to determine query equivalence under database constrains. Recently the chase procedure has experienced a revival due to its application in data exchange. In this chapter we review the chase algorithm and its properties as well as its application in data exchange.

**1998 ACM Subject Classification** H.2.5 [Heterogeneous Databases]: Data translation

**Keywords and phrases** chase, chase termination, data exchange, incomplete information

**Digital Object Identifier** 10.4230/DFU.Vol5.10452.1

## 1 Introduction

The main focus of this chapter is an introduction to the chase procedure and its importance in data exchange, as it is already announced in the title. A retrospective look at the evolution of the chase procedure proves with no doubt its importance and effectiveness in solving several data related problems. Originally, the chase was developed for testing logical implication between sets of embedded dependencies [36]. In fact, the logical implication problem tests whether all databases satisfying a set of dependencies must also satisfy another given dependency. Later, the chase was reformulated for other types of dependencies such as functional, join and multivalued dependencies [37, 49]. Beeri and Vardi [10] proposed a unified treatment for the implication problem by introducing the chase for tuple-generating and equality-generating dependencies, classes of dependencies large enough to express all the previous classes. Moreover, the chase procedure was also shown to be useful for determining if two database instances (that may contain nulls) represent the same set of possible instances under a set of dependencies [43]. Finally, the chase was also used for testing query equivalence and containment under database constraints [3, 31].

More recently, the chase procedure has gained a lot of attention due to its usefulness in: data integration [33, 11], ontologies [14, 13], inconsistent databases and data repairs [5, 2, 23], data exchange [19], query optimization [17, 42], peer data exchange [9], and data correspondence [23]. In this chapter we will focus on the advantages of using the chase procedure in data exchange. We will show that the chase can be used to compute representative target solutions in data exchange. Intuitively, the data exchange problem consists of transforming a source database into a target one according to a set of source to target dependencies describing the mapping between the source and the target. The set of dependencies may also include target dependencies, that is constraints over the target database. It is important to mention that the source and the target schemata are considered to be distinct. To be more precise: given a source instance $I$ and a set $\Sigma$ of source-to-target and target dependencies, an instance $J$ over the target schema is said to be a data-exchange solution (or simply solution) for $I$ and $\Sigma$, if $I \cup J$ satisfies all dependencies in $\Sigma$. One of the most important representation for this (usually infinite) set of solutions was introduced by Fagin et al. [19]. They considered the finite instance obtained by chasing the initial source instance with the set of dependencies. Such an instance, if it exists, was called a *universal solution*. In their paper, Fagin et al. showed that the universal solution is a good candidate to be materialized on the target. In particular, the universal solution can be used to compute certain answers to (unions of) conjunctive queries over the target instance.

Even though not exhaustive, this chapter investigates the chase procedure when applied against a set of tuple-generating and equality generating dependencies. We also investigate different chase variations proposed for data exchange and review their properties. Section 3 is devoted to the chase procedure and to some of its variations when the constraints are specified by sets of tuple-generating and equality generating dependencies. This section it is not only focused on mapping constraints (i.e. source-to-target and target dependencies) but also general constraints giving us a full picture for the chase termination problem. In the same Section 3, we will also see that each of the chase variation presented computes instances that are homomorphically equivalent and thus any of this chase variations can be used in computing universal solutions. Even more, to the best of our knowledge, there exists only one chase variation which is complete in finding universal solutions. It is known that the chase procedure might not terminate for some input instances; even more, it was shown that it is undecidable to test if a chase procedure terminates for a given set of dependencies and a given input instance. Given this, there was tremendous work in finding classes of dependencies that ensure the chase termination for all input instances. Section 4 presents some of the main such classes and reviews the subset relationship and complexity of testing the membership problem for these classes. In Section 5 we review the role played by the chase procedure in data exchange. Finally, Section 6 presents an extension of the chase process that deals with larger classes of dependencies including inequalities, disjunctions and negations. All the proofs presented in this chapter are only sketches, the complete proofs can be found in the mentioned literature.

## 2 Preliminaries

For basic definitions and concepts we refer to [1]. We will consider the complexity classes PTIME, NP, coNP, DP, RE, coRE, and first few levels of the polynomial hierarchy. For definitions of these classes we refer to [45].

Let us start with some preliminary notions. A *schema* $\mathbf{R}$ is a finite set $\{R_1, \ldots, R_n\}$ of relation names, each $R_i$ having a fixed arity, $arity(R_i)$. Let Const be a countably infinite set of constants, Null be a countably infinite set of labeled nulls and Var be a countable infinite set of variables, such that the sets are pairwise disjoint. From the domain $\mathsf{Dom} = \mathsf{Const} \cup \mathsf{Null}$ and the finite set $\mathbf{R}$ we build up a Herbrand structure consisting of all expressions of the form $R(a_1, a_2, \ldots, a_k)$, where $R$ is a $k$-ary relation name from $\mathbf{R}$ and $a_i$'s are values in Dom. Such an expression is called a tuple. A database instance $I$ is then simply a finite set of tuples. We denote the set of values occurring in an instance $I$ by $dom(I)$. An instance $I$, such that $dom(I) \subseteq \mathsf{Const}$, is called a *ground instance*.

Let then $I$ and $J$ be instances over a schema $\mathbf{R}$. A *homomorphism* $h$ from $I$ to $J$ is a function on $\mathsf{Const} \cup \mathsf{Null}$, that is identity on Const, extended to tuples, relations and instances in the natural way, such that $h(I) \subseteq J$. We write $I \to J$ in case there exists a homomorphism from $I$ to $J$. By $I \leftrightarrow J$ we denote the fact that $I \to J$ and $J \to I$. A homomorphism from $I$

to $J$ is said to be *full* if $h(I) = J$. A full injective homomorphism is called *embedding.* A homomorphism $h$ from $I$ to $J$ is said to be a *retraction* if $h$ is identity on $dom(J)$. In this case $J$ is called a *retract* of $I$. An instance $J$ is said to be a proper *retract* of instance $I$, if $J$ is a retract of $I$ and $J \subset I$. An instance $I$ is said to be a core if it does not have any proper retract. An instance $J$ is said to be a core of $I$ if $J$ is a retract of $I$ and it is also a core. The cores of an instance $I$ are unique up to isomorphism and therefore we can talk about *the core* of an instance $I$ and denote it $core(I)$.

A relational atom is an expression of the form $R(\bar{x})$, where $R$ is a relational symbol from a schema $\mathbf{R}$ and $\bar{x} \in (\mathsf{Const} \cup \mathsf{Var})^{arity(R)}$. For an easier representation, by $\bar{x}$ we also represent the sets of elements in $\bar{x}$ and we denote by $|\bar{x}|$ the cardinality of such set. A conjunctive query $\varphi(\bar{x})$ over a schema $\mathbf{R}$ is a conjunction of relational atoms from $\mathbf{R}$, where $\bar{x}$ denotes the variables of the atoms in $\varphi$. $\mathsf{CQ}$ identifies the class of conjunctive queries and $\mathsf{UCQ}$ the class of unions of conjunctive queries. The class $\mathsf{CQ}^{\neq}$ denotes all conjunctive queries that also allow the inequality atom (similarly is defined the class $\mathsf{UCQ}^{\neq}$). The extension of previous classes by allowing negation gives $\mathsf{CQ}^{\neg}$,$\mathsf{UCQ}^{\neg}$, $\mathsf{CQ}^{\neg,\neq}$ and $\mathsf{UCQ}^{\neg,\neq}$. Given a formula $\alpha: R_1(\bar{x}_1) \wedge R_2(\bar{x}_2) \wedge \ldots \wedge R_n(\bar{x}_n)$ and a mapping $h : (\mathsf{Const} \cup \mathsf{Var}) \to (\mathsf{Const} \cup \mathsf{Null})$, identity on $\mathsf{Const}$, by $h(\alpha)$ we denote the instance $\{R_1(h(\bar{x}_1)), R_2(h(\bar{x}_2)), \ldots, R_n(h(\bar{x}_n))\}$.

Finally, a *tuple generating dependency* (tgd) is a first order sentence $\xi$ of the form: $\forall \bar{x}, \bar{y}\ \big(\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})\big)$, where $\alpha$ (the body) and $\beta$ (the head) are conjunctive queries, $\bar{x}$ and $\bar{y}$ denote the universally quantified variables, and $\bar{z}$ the existentially quantified ones. We denote by $body(\xi)$ the set of all atoms in the body and by $head(\xi)$ the set of all atoms in the head. An *equality generating dependency* (egd) is a first order sentence $\xi$ of the form $\forall \bar{x}\ \big(\alpha(\bar{x}) \to x_1 = x_2\big)$. An egd is like a tgd, except that the consequent is an equality between the variables $x_1$ and $x_2$ that also are part of $\bar{x}$. For simplicity for these types of formulae, we will omit the universal quantifiers; also the conjunction between atoms will be denoted by comma. Thus, the tgd $\forall x, y\ \big(R(x, y) \wedge R(y, x) \to \exists z\ T(x, z) \wedge S(z)\big)$ will be simply denoted as $R(x, y), R(y, x) \to \exists z\ T(x, z), S(z)$. A *full tgd* is a tgd that has no existentially quantified variables. A LAV tgd is a tgd with only one atom in the body. Let $\xi$ be a tgd of the form $\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$, and $\bar{a} \in (\mathsf{Const})^{|\bar{x}|+|\bar{y}|}$. We denote by $\xi(\bar{a})$ the first order sentence obtained from $\xi$ by replacing the universal quantified variables with the corresponding constants in $\bar{a}$. An instance $I$ is said to *satisfy* a set of dependencies, denoted $I \models \Sigma$, if $I$ satisfies $\Sigma$ in the theoretic standard model sense.

## 3 The chase procedure

The importance of the chase procedure in data exchange was first brought to the forefront by Fagin, Kolaitis, Miller and Popa in their seminal paper [19], where the chase was used as a tool for constructing a "general" solution to the data-exchange problem. In their approach the chase procedure applies at each iteration a chase step, that either adds a new tuple or changes the instance to model some equality generating dependency, or fails when the instance could not be changed to satisfy an equality generating dependency. Based on this chase procedure, several variations were proposed [12, 16, 38, 24]. To differentiate them, we will call the chase procedure presented by Fagin et al. [19] *the standard chase.* Since most of the practical database constraints (such as key and inclusion dependencies) can be represented as sets of tuple generating (tgd) and equality generating (egd) dependencies, in this section we will present the chase procedure applied on such dependencies. Later on, more precisely in Section 6, we will introduce a variation of the chase procedure that also deals with dependencies containing inequalities and disjunctions. For an ease of notation, through this section if not mentioned otherwise, we will use the notation $I$ to represent an arbitrary instance over a given schema and $\Sigma$ to refer to an arbitrary set of tgds and egds over a schema explicitly mentioned if it does not follow directly from the context.

## 3.1 The chase step

The chase procedure is a repetitive application of a chase step. Each chase step "applies" a tgd or egd, on a subset of the instance.

**The tgd chase step.** Let $I$ be an instance and $\xi$ be the tgd $\alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z})$ both over a schema $\mathbf{R}$. A pair $(\xi, h)$ is said to be a *trigger* for $I$, if $h$ is a homomorphism such that $h(\alpha(\bar{x},\bar{y})) \subseteq I$. In case we also have that there is no extension $\tilde{h}$ of $h$ such that $\tilde{h}(\beta(\bar{x},\bar{z})) \subseteq I$, then $(\xi, h)$ is said to be an *active trigger* for $I$. The tgd $\xi$ is said to be *applicable* to $I$ with homomorphism $h$ if $(\xi, h)$ is a trigger (active or not) for $I$.

To *fire* the trigger $(\xi, h)$ means to transform $I$ into the instance $J = I \cup \tilde{h}(\beta(\bar{x},\bar{z}))$, where $\tilde{h}$ is a *distinct extension* of $h$, i.e. an extension of $h$ that assigns new fresh nulls to the existential variables in $\beta$. By "new fresh" we mean the next unused element in some fixed enumeration of the nulls. We call this transformation as an *oblivious-chase step* and denote it $I \xrightarrow{*,(\xi,h)} J$. In case $(\xi, h)$ is an active trigger for $I$, the transformation is called *standard-chase step* and is denoted $I \xrightarrow{(\xi,h)} J$. Clearly any standard-chase step is also an oblivious-chase step but the converse does not always hold.

▶ **Example 1.** Let us consider instance $I = \{R(a,b), R(b,a), S(b,c)\}$ and tgd $\xi$:

$$R(x,y), R(y,x) \to \exists z\ S(x,z).$$

Homomorphism $h = \{x/a, y/b\}$ maps the body of $\xi$ to $I$ and there is no extension of $h$ that maps the head of $\xi$ into $I$. That is, the trigger $(\xi, h)$ is active for $I$ and $I \xrightarrow{(\xi,h)} J$, where $J = I \cup \{S(a,X)\}$ and $\tilde{h}(z) = X$. On the other hand, for the homomorphism $h' = \{x/b, y/a\}$ the pair $(\xi, h')$ is a trigger for $I$, but it is not an active trigger. In this case $I \xrightarrow{*,(\xi,h')} J'$, where $J' = I \cup \{S(b,X)\}$.

The complexity of testing if there exists a trigger (active trigger) for a given instance $I$ and a fixed (or given) tgd $\xi$ is given by the following theorem:

▶ **Theorem 2.** [26] *Let $\xi$ be a tgd and $I$ an instance. Then*
1. *for a fixed $\xi$, testing whether there exists a trigger or an active trigger on a given $I$ is polynomial;*
2. *testing whether there exists a trigger for a given $\xi$ on a given $I$ is* NP*-complete;*
3. *testing whether there exists an active trigger for a given $\xi$ and a given $I$ is $\Sigma_2^p$-complete.*

**Proof.** Let us consider $\xi$ to be a tgd of the form $\alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{z})$. The polynomial cases can be verified by checking all homomorphisms from the body of the dependency into the instance. We also need to consider for the active trigger problem if it has for each such homomorphism an extension that maps the head of the dependency into the instance. These tasks can be carried out in $O(n^{|\alpha|})$ and $O(n^{|\alpha|+|\beta|})$ time, respectively.

It is easy to see that the trigger existence problem is NP-complete in combined complexity, as the problem is equivalent to testing whether there exists a homomorphism between two instances (in our case $\alpha$ and $I$); a problem known to be NP-complete.

Regarding the combined complexity of the active trigger existence problem, we observe that it is in $\Sigma_2^P$, since one may guess a homomorphism $h$ from $\alpha$ into $I$, and then use an NP oracle to verify that there is no extension $h'$ of $h$, such that $h'(\beta) \subseteq I$. In the case of the lower bound, we will reduce the following problem to the active trigger existence problem. Let $\phi(\bar{x}, \bar{y})$ be a Boolean formula in 3CNF over the variables in $\bar{x}$ and $\bar{y}$. *Is the formula* $\exists\bar{x}\,\neg\big(\exists\bar{y}\,\phi(\bar{x},\bar{y})\big)$ *true?* The problem is a variation of the standard $\exists\forall$-QBF problem [48] and known to be $\Sigma_2^P$-complete [46].

For the reduction, let $\phi$ be given. We construct an instance $I_\phi$ and a tgd $\xi_\phi$. The instance $I_\phi$ is:

| $F$ | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

| $N$ | |
|---|---|
| 0 | 1 |
| 1 | 0 |

The tgd $\xi_\phi = \alpha \to \beta$ is constructed as follows: for each variable $x \in \bar{x}$ in $\phi(\bar{x},\bar{y})$, the body $\alpha$ will contain the atom $N(x,x')$ ($x'$ is used to represent $\neg x$). The head $\beta$ is existentially quantified over that set $\bigcup_{y\in\bar{y}}\{y,y'\}$ of variables. For each conjunct $C$ of $\phi$, we place an atom $F(x,y,z)$ in $\beta$, where $x, y$ and $z$ are the variables in $C$, with the convention that if the variable $x$ is negated in $C$, then $x'$ is used in the atom. Finally, for each $y \in \bar{y}$, we place in $\beta$ the atom $N(y,y')$, denoting that $y$ and $y'$ should not have the same truth assignment. It is easy to note that $\exists\bar{x}\,\neg\big(\exists\bar{y}\,\phi(\bar{x},\bar{y})\big)$ is true, if and only if there exists an active trigger $(\xi_\phi, h)$ for $I_\phi$. ◀

**The egd chase step.** Let $I$ be an instance and $\xi$ the egd $\alpha(\bar{x}) \to x_i = x_j$, where $x_i, x_j \in \bar{x}$. We say that $\xi$ is *applicable* to $I$ with the homomorphism $h$, if the following holds:

1. $h$ maps the atoms of $\alpha(\bar{x})$ to tuples of $I$,
2. $h(x_i) \neq h(x_j)$.

The pair $(\xi, h)$ is called an egd active trigger for $I$, or simply a trigger. Let $(\xi, h)$ be an egd trigger for $I$. In case $h$ maps both variables $x_i$ and $x_j$ to constants, then we say that the egd chase step *fails*, and represent this by $I \xrightarrow{(\xi,h)} \perp$. Otherwise, we say that the egd chase step *does not fail* and denote this by $I \xrightarrow{(\xi,h)} J$, where instance $J$ is computed as follows:

1. if both $h(x_i)$ and $h(x_j)$ are labeled nulls, then $J$ is obtained from $I$ by replacing all occurrences of $h(x_i)$ with $h(x_j)$, considering that there is an enumeration of the variables such that $i < j$.
2. if either $h(x_i)$ or $h(x_j)$ is a constant and the other is a labeled null, then $J$ is obtained from $I$ by replacing all occurrences of the labeled null with the constant.

▶ **Example 3.** Consider instance $I = \{R(a,b), R(c,X), R(X,Y)\}$ and $\xi$: $R(x,y) \to x = y$. There are three distinct homomorphisms that map the body of $\xi$ into $I$: $h_1 = \{x/a, y/b\}$, $h_2 = \{x/c, y/X\}$ and $h_3 = \{x/X, y/Y\}$. As $h_1(x)$ and $h_1(y)$ are distinct constants, it follows that $I \xrightarrow{(\xi,h_1)} \perp$. On the other hand, $h_2(x)$ is a constant and $h_2(y)$ is a null. Thus, we have $I \xrightarrow{(\xi,h_2)} J$, where $J = \{R(a,b), R(c,c), R(c,Y)\}$ is obtained by replacing all occurrences of null $X$ with constant $c$ in $I$. Finally, $h_3$ maps both variables $x$ and $y$ to distinct nulls, making the egd $\xi$ applicable on $I$ with the homomorphism $h_3$. Hence $I \xrightarrow{(\xi,h_3)} J'$, where $J' = \{R(a,b), R(c,Y), R(Y,Y)\}$ is obtained from $I$ by replacing $X$ with $Y$, considering that $y$ follows $x$ in the variable enumeration.

Similarly to the tgd chase step, we have the following complexity results for the egd trigger-existence problem. Note that for egds the trigger and active trigger notions coincide.

▶ **Theorem 4.** *Let $\xi$ be an egd and $I$ an instance. Then*

1. *for a fixed $\xi$, testing whether there exists a trigger on a given $I$ is polynomial, and*
2. *testing whether there exists a trigger for a given $\xi$ on a given $I$ is* NP*-complete.*

**Proof.** Similar with the proof of Theorem 2. ◀

## 3.2 The chase algorithm

Using the previously introduced chase steps, we are now ready to present the standard-chase algorithm. This algorithm can be described as an iterative application of the standard-chase steps. In case one of the egd chase steps fails, then the algorithm will fail. If the algorithm does not fail, it nondeterministically chooses another active trigger, tgd or egd, and proceeds with the corresponding standard-chase step. The algorithm terminates when either one of the egd chase step fails or when there are no other active triggers. More formally, the standard-chase algorithm can be described as follows:

```
STANDARD-CHASE(I,Σ)
1 $I_0 := I$; $i := 0$;
2 if exists active trigger $(\xi, h)$ for $I_i$
3     then
4           if $I_i \xrightarrow{(\xi,h)} \perp$
5              then return FAIL
6              else $I_i \xrightarrow{(\xi,h)} I_{i+1}$; $i := i+1$
7     else return $I_i$
8 goto 2
```

Note that the previous algorithm introduces a nondeterministic step at line 2, induced by the trigger choice. This makes the chase process to be viewed as a tree, where level $i$ in the tree represents the $i$-th step in the chase algorithm, and where to each node a new edge is added for each of the applicable active trigger. Each path from the root of the tree to a leaf node represents an *execution branch*, or simply a branch, similarly to the nondeterministic finite automata. Thus the algorithm may return different instances depending on the considered branch. There are cases when, for some branches, the algorithm fails while it does not fail for other branches, as it is shown in Example 6. This happens by exhaustively choosing the same dependencies in the nondeterministic step.

Moreover, the standard-chase algorithm stops if it either fails, due to an egd trigger at step 4, or there are no other active triggers to be applied. As the tgds are adding new tuples to the instance, it may be that the chase algorithm never terminates as in Example 5.

Fagin et al. [19] showed that in case the standard-chase algorithm fails on one execution branch, then it will fail on all finite branches.

For the branches for which the algorithm does not fail, a *standard-chase sequence* is a finite or infinite sequence $(I_0, I_1, I_2, \ldots, I_n, \ldots)$ such that $I_0 = I$ and $I_i \xrightarrow{(\xi,h)} I_{i+1}$, for some $i \geq 0$ and some active trigger $(\xi, h)$. If for some branch the algorithm terminates in the finite, then there exists a positive integer $n$ such that for the standard-chase sequence $(I_0, I_1, I_2, \dots, I_n)$ there is no active trigger for $I_n$.

As shown in the following example, a standard chase sequence may be finite or infinite, for the same set of tgds and the for same input instance.

▶ **Example 5.** Consider instance $I = \{R(a,b)\}$ and tgds:

$$\begin{aligned} \xi_1 &= R(x,y) \rightarrow R(y,x), \text{ and} \\ \xi_2 &= R(x,y) \rightarrow \exists z\, R(y,z). \end{aligned}$$

If first we chose the tgd trigger $(\xi_1, \{x/a, y/b\})$, the tuple $R(b,a)$ is added to the instance $I$ forming an instance $I' = I \cup \{R(b,a)\}$. It can be easily noticed that there is no other active trigger on $I'$ involving either $\xi_1$ or $\xi_2$. From this it follows that the sequence $(I, I')$ is a finite standard-chase sequence. On the other hand, if in the standard-chase algorithm we chose first the active trigger $(\xi_2, \{x/a, y/b\})$, and from there on only chose active triggers over $\xi_2$, we get the following infinite chase sequence:

$$\begin{array}{ccccccccc} I_0 = I & \xrightarrow{(\xi_2,h_1)} & I_1 & \xrightarrow{(\xi_2,h_2)} & \dots & \xrightarrow{(\xi_2,h_n)} & I_n & \xrightarrow{(\xi_2,h_{n+1})} & \dots \\ \begin{array}{cc} \multicolumn{2}{c}{R} \\ \hline a & b \end{array} & & \begin{array}{cc} \multicolumn{2}{c}{R} \\ \hline a & b \\ b & X_1 \end{array} & & & & \begin{array}{cc} \multicolumn{2}{c}{R} \\ \hline a & b \\ b & X_1 \\ X_1 & X_2 \\ \multicolumn{2}{c}{\dots} \\ X_{n-1} & X_n \end{array} & & \end{array}$$

The next example shows a case when the standard-chase algorithm fails on some branches and does not terminate (implicitly does not fail) on others.

▶ **Example 6.** Let us now consider a slightly changed set of dependencies from the previous example:

$$\begin{aligned} \xi_1 &= R(x,y) \rightarrow T(y,x); \\ \xi_2 &= T(x,y) \rightarrow x = y; \text{ and} \\ \xi_3 &= R(x,y) \rightarrow \exists z\, R(y,z). \end{aligned}$$

Consider the instance $I = \{R(a,b)\}$. If applying an active trigger $(\xi_1, \{x/a, y/b\})$, it will add the tuple $T(a,b)$ to $I$. Next, when applying the active trigger $(\xi_2, \{x/a, y/b\})$ the standard-chase algorithm will fail. However, if the chosen branch uses only the triggers over $\xi_3$, the standard-chase algorithm will not terminate, as previously shown.

In the previous example the standard-chase algorithm did not fail because we exhaustively applied active triggers over the same dependency. To avoid such cases, the chase algorithm is required to be *fair*, defined as follows:

▶ **Definition 7.** Let $I_0$ be an instance and $\Sigma$ a set of tgds and egds. A standard-chase sequence $(I_0, I_1, \dots)$ is said to be *fair* if for all $i$ and for all active triggers $(\xi, h)$ for $I_i$, where $\xi \in \Sigma$, there exists $j$ such that either $I_j \xrightarrow{(\xi,h)} I_{j+1}$ or the trigger $(\xi, h)$ is not active for $I_j$. A standard-chase algorithm is said to be *fair*, if it only produces fair chase sequences.

In the rest of this chapter we will consider, if not mentioned otherwise, all the chase algorithms to be fair.

Let us now turn our attention to standard-chase algorithms that terminate in a finite number of steps. The following proposition shows the relationship between the finite instances returned by the algorithm.

▶ Proposition 8. [19] If $K$ and $J$ are two finite instances returned by the standard-chase algorithm on two distinct execution branches, with input $I$ and $\Sigma$, then $K$ and $J$ are homomorphically equivalent, that is $K \leftrightarrow J$.

Based on the homomorphic equivalence class, if there exists an execution branch for which the standard-chase algorithm with input $I$ and $\Sigma$ terminates in the finite and does not fail, then we denote by $chase^{\mathsf{std}}{}_{\Sigma}(I)$ one representative of the equivalence class for the resulting finite instances. If the standard chase fails or if it does not terminate in the finite on all branches, then we set $chase^{\mathsf{std}}{}_{\Sigma}(I) = \perp$. With the instance $I$ and the dependencies $\Sigma$ defined in Example 5, we have $chase^{\mathsf{std}}{}_{\Sigma}(I) = \{R(a,b), R(b,a)\}$. On the other hand, with the input instance and the dependencies defined in Example 6 we have $chase^{\mathsf{std}}{}_{\Sigma}(I) = \perp$. The following theorem, developed by Fagin et. al, states one of the main properties of the standard-chase algorithm. As seen later in this section, this property also holds for the other chase variations.

▶ **Theorem 9.** [19] *If* $chase^{\mathsf{std}}{}_{\Sigma}(I) \neq \perp$, *then* $chase^{\mathsf{std}}{}_{\Sigma}(I) \models \Sigma$ *and* $I \rightarrow chase^{\mathsf{std}}{}_{\Sigma}(I)$.

Let us now turn our attention to the problem defined as the termination of standard-chase algorithm. It is easy to see that the cause of non-termination lies in the existentially quantified variables in the head of tgds. Thus, for simplicity, for the following classes we omitted egds.

▶ **Definition 10.** Given an instance $I$, by $\mathsf{CT}^{\mathsf{std}}_{I,\forall}$ we denote the class of tgd sets such that $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{I,\forall}$ iff all standard-chase sequences for $I$ and $\Sigma$ are finite. We denote by $\mathsf{CT}^{\mathsf{std}}_{I,\exists}$ the class of tgd sets such that $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{I,\exists}$ iff there exist some standard-chase sequences for $I$ and $\Sigma$ that are finite.

The previous notations are extended to classes of tgd sets for which the standard chase terminates on all input instances as follows:

▶ **Definition 11.** We denote by $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ the class of tgd sets such that $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ iff for all instances $I$ all standard-chase sequences of $I$ with $\Sigma$ are finite. We denote by $\mathsf{CT}^{\mathsf{std}}_{\forall\exists}$ the class of tgd sets such that $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$ iff for all instances $I$ there exists at least one standard-chase sequence of $I$ and $\Sigma$ that is finite.

From the termination classes definition it is clear that $\mathsf{CT}^{\mathsf{std}}_{\forall\forall} \subseteq \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$. Also from the set of dependencies presented in Example 5, it follows that the inclusion is strict.

Deutsch, Nash and Remmel in [16] showed that, given $I$ and $\Sigma$, the problems of testing whether $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{I,\forall}$ or $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{I,\exists}$ are undecidable in general. More recently, Grahne and O. [26] extended this undecidability result to the $\mathsf{CT}^{\mathsf{std}}_{\forall\exists}$ class too. That is for a given $\Sigma$, the problem of testing if $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$ is coRE-complete. In case we allow a single *denial constraint*, then the class $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ is coRE-complete as well. Where a denial constraint is a tgd of the form $\alpha(\bar{x}) \rightarrow \perp$, which is satisfied by an instance $I$ only if there is no homomorphism $h$ such that $h(\alpha(\bar{a})) \subseteq I$,

Given the previous result, the next best hope is to find large decidable classes of dependencies included in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. One such class is the one of full tgds, that is tgds without existential quantifiers. In Section 4 we review other decidable classes of dependency sets that are known to be in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.

Before ending this section we need to reiterate that the termination classes are defined over tgds. Even if the egds do not introduce new nulls, they still may play an important role in the standard-chase termination problem, as shown in the next example.

▶ **Example 12.** Let $\Sigma = \{\xi_1, \xi_2, \xi_3\}$, where:

$$\begin{aligned} \xi_1 &= R(x,y) \to \exists z\; S(y,z); \\ \xi_2 &= S(x,x) \to \exists z\; R(x,z);\ \text{and} \\ \xi_3 &= S(x,y) \to x = y. \end{aligned}$$

Let $I = \{R(a,b)\}$. It is easy to see that the standard-chase algorithm converges to the infinite instance $J = \{R(a,b), R(b,X_1), \ldots, R(X_{n-1},X_n), \ldots, S(b,b), S(X_1,X_1), \ldots, S(X_n,X_n), \ldots\}$. On the other hand, $\Sigma' = \{\xi_1, \xi_2\} \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, that is without the egd $\xi_3$ the standard chase algorithm terminates on all execution branches with any input instance.

## 3.3 Chase variations

After the standard chase was presented as a method of computing "general" solutions in data exchange [19], many variations of the standard-chase algorithm were proposed in the literature [12, 16, 38, 24] . In the remaining part of this section, dedicated to the chase algorithm, we try to differentiate between the main chase variations by highlighting their termination properties.

### 3.3.1 The oblivious chase

This focuses on one of the simplest variations of the standard chase named the oblivious chase (also known as naïve chase). This procedure is based on the relaxation of the chase step. The oblivious chase presented here differs from the one described by Cali et al. [12] by not relying on any order. As we will see, this does not affect the finite instance returned by the chase algorithm.

The oblivious-chase algorithm is an iterative application of the oblivious-chase step, that is at each iteration all triggers are considered, and not only the active ones as in the standard-chase algorithm. Recall that for the trigger $(\xi, h)$ and the instance $I$ the oblivious-chase step is denoted as $I \xrightarrow{*,(\xi,h)} J$, where instance $J$ is constructed the same way as in the standard-chase step. Note that if $(\xi, h)$ is a trigger for $I$, then $(\xi, h)$ will also be a trigger for $J$, where $I \xrightarrow{*,(\xi,h)} J$. To avoid such infinite loops, the oblivious-chase algorithm applies each trigger only once.

▶ **Example 13.** Consider the instance $I = \{R(a,b), R(b,a), S(b,c)\}$ and the tgd $\xi$ defined as $R(x,y), R(y,x) \to \exists z\; S(x,z)$ as in Example 1. The homomorphism $h = \{x/a, y/b\}$ maps the body of $\xi$ to $I$, and there is no extension of $h$ that maps the head of $\xi$ into $I$. This makes $(\xi, h)$ both a standard and an oblivious-chase trigger. However, the homomorphism $h_1 = \{x/b, y/a\}$ also maps the body of $\xi$ to $I$, but there exists the extension $\tilde{h}_1 = \{x/b, y/a, z/c\}$ of $h_1$, such that $\tilde{h}_1$ maps the head of $\xi$ into $I$. Hence $(\xi, h_1)$ is a trigger but not an active trigger for $I$. The instance $J$ is obtained by applying the oblivious-chase step $I \xrightarrow{*,(\xi,h_1)} J$, where $J = I \cup \{S(b,Y)\}$ and $Y$ is a new labeled null.

Because of the nondeterministic way the oblivious-chase algorithm selects the triggers at each iteration, we may have different execution branches. Similarly to the termination classes defined for the standard chase, we introduce corresponding termination classes of tgd sets for the oblivious chase: $\mathsf{CT}^{\mathsf{obl}}_{I,\exists}$, $\mathsf{CT}^{\mathsf{obl}}_{I,\forall}$, $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ and $\mathsf{CT}^{\mathsf{obl}}_{\forall\exists}$.

The following proposition shows that the termination classes are not affected by the nondeterministic nature of the algorithm.

▶ Proposition 14. Let $I$ be an instance. Then $\mathsf{CT}^{\mathsf{obl}}_{I,\forall} = \mathsf{CT}^{\mathsf{obl}}_{I,\exists}$ and $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} = \mathsf{CT}^{\mathsf{obl}}_{\forall\exists}$.

**Proof.** The proof follows from the observation that for the oblivious-chase algorithm, when the input set of dependencies are only tgds, the set of triggers applied on each branch is the same, up to isomorphism. Thus, if the oblivious chase terminates on one execution branch, then it will terminate on all branches. ◀

From the previous proof it also follows that in case the oblivious chase terminates for instance $I$ and set of tgds $\Sigma$, then the returned on all execution branches are isomorphically equivalent. As we will see in the following example, if we allow egds, the instances returned are not guaranteed to be isomorphically equivalent. Still, using the same proof techniques as the one used to prove Theorem 9, it can be shown that in this case, if the chase terminates and does not fail, the instances returned are homomorphically equivalent. Thus, if the oblivious chase terminates with input $I$ and $\Sigma$ (containing both tgds and egds), then we denote by $chase^{\mathbf{obl}}{}_{\Sigma}(I)$ one representative instance of the homomorphic equivalence class. If the oblivious chase fails or if it does not terminate, we set $chase^{\mathbf{obl}}{}_{\Sigma}(I) = \bot$.

▶ **Example 15.** Let $\Sigma = \{\xi_1, \xi_2, \xi_3\}$, where:

$$\begin{aligned} \xi_1 &= S(x) \to \exists y\ R(x,y); \\ \xi_2 &= R(x,y) \to \exists z\ T(y,z);\ \text{and} \\ \xi_3 &= R(x,y) \to x = y. \end{aligned}$$

Let us now consider the instance $I = \{S(a)\}$. If we apply the dependencies in the order $\xi_1$, $\xi_2$, $\xi_3$ and then $\xi_2$ again, we get the following instance $J_0 = \{S(a), R(a,a), T(a,X_1), T(a,X_2)\}$. But, if we apply the dependencies in the order $\xi_1$, $\xi_3$ and finally $\xi_2$, the instance returned by the oblivious-chase algorithm is $J_1 = \{S(a), R(a,a), T(a,Y_1)\}$. Clearly $J_0$ and $J_1$ are not isomorphically equivalent but they are homomorphically equivalent.

From the observation that all active triggers are also triggers, it follows that:

▶ Proposition 16. $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.

**Proof.** The inclusion follows directly from the definition of the trigger and the active trigger. For the strict inclusion part consider dependency set from Example 17. ◀

▶ **Example 17.** Consider $\Sigma = \{R(x,y) \to \exists z\ R(x,z)\}$. Clearly there is no active trigger on $\Sigma$ for any instance $I$. On the other hand, for $I = \{R(a,b)\}$ the oblivious-chase algorithm will create the following infinite chase sequence:

$$\begin{array}{c} I_0 = I \\ \begin{array}{cc} \multicolumn{2}{c}{R} \\ \hline a & b \end{array} \end{array} \xrightarrow{*,(\xi,h_1)} \begin{array}{c} I_1 \\ \begin{array}{cc} \multicolumn{2}{c}{R} \\ \hline a & b \\ a & X_1 \end{array} \end{array} \xrightarrow{*,(\xi,h_2)} \dots \xrightarrow{*,(\xi,h_n)} \begin{array}{c} I_n \\ \begin{array}{cc} \multicolumn{2}{c}{R} \\ \hline a & b \\ a & X_1 \\ a & X_2 \\ \multicolumn{2}{c}{\dots} \\ a & X_n \end{array} \end{array} \xrightarrow{*,(\xi,h_{n+1})} \dots$$

In order to relate termination of the standard and the oblivious algorithms, we introduce a transformation called *enrichment* that takes a tgd $\xi = \alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z})$ over a schema $\bar{R}$, and converts it into the tgd $\hat{\xi} = \alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z}), H(\bar{x},\bar{y})$, where $H$ is a new relational symbol that does not appear in $\mathbf{R}$. For a set $\Sigma$ of tgds defined on schema $\mathbf{R}$, the transformed set is $\widehat{\Sigma} = \{\hat{\xi} : \xi \in \Sigma\}$. Using the enrichment notion, we can present the relation between the standard and oblivious-chase terminations.

▶ **Theorem 18.** [25] *Let $\Sigma$ be a set of tgds and $I$ an instance. Then we have:*
1. $\Sigma \in \mathsf{CT}^{\mathsf{obl}}_{I,\forall}$ *if and only if* $\widehat{\Sigma} \in \mathsf{CT}^{\mathsf{std}}_{I,\forall}$, *and*
2. $\Sigma \in \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ *if and only if* $\widehat{\Sigma} \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.

**Proof.** It follows from the observation that for any instance $I$ if $(\xi, h)$ is a trigger for $I$, then $(\widehat{\xi}, h)$ is also an active trigger for $I$. ◀

Cali et al. showed in [12] that it is undecidable if the oblivious chase terminates for a given input $I$ and given set of tgds $\Sigma$. The same result applies for the class $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ if we allow at least one denial constraint [26]. It remains an open problem if the class $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ remains undecidable when considering only sets of tgds.

We close the presentation of the oblivious-chase algorithm by linking together the finite instances resulting from both chase algorithms.

▶ **Theorem 19.** [12] *Let $I$ be an instance and let $\Sigma$ be a set of tgds and egds, such that $chase^{\mathbf{obl}}{}_{\Sigma}(I) \neq \bot$. Then $chase^{\mathbf{std}}{}_{\Sigma}(I) \leftrightarrow chase^{\mathbf{obl}}{}_{\Sigma}(I)$ and $chase^{\mathbf{obl}}{}_{\Sigma}(I) \models \Sigma$.*

### 3.3.2 The semi-oblivious chase

The semi-oblivious-chase method was first introduced by Marnette in [38]. For this, let $\xi$ be a tgd $\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$; then the triggers $(\xi, h)$ and $(\xi, g)$ are considered equivalent if $h(\bar{x}) = g(\bar{x})$. The semi-oblivious chase works as the oblivious one, except that exactly one trigger from each such equivalence class is fired in a branch.

For a better differentiation between the chase algorithms presented so far consider the following example:

▶ **Example 20.** Let $\Sigma = \{\xi\}$ contain the tgd $\forall x, y\ R(x, y) \to \exists z\ T(x, z)$, and consider the instance $I = \{R(a, b), R(a, c), R(d, e), T(a, a)\}$. In this case there exist only three triggers $\tau_1 = (\xi, \{x/a, y/b\})$, $\tau_2 = (\xi, \{x/a, y/c\})$ and $\tau_3 = (\xi, \{x/d, y/e\})$. From these triggers only $\tau_3$ is an active trigger for $I$. Thus, there is only one step to be executed for the standard chase: $I \xrightarrow{\tau_3} J^{\mathbf{std}}$, where $J^{\mathbf{std}} = I \cup \{T(d, X_1)\}$. Because the homomorphism from the trigger $\tau_1$ maps $x$ to value $a$ as the homomorphism from the trigger $\tau_2$, it follows that $\tau_1$ is equivalent with $\tau_2$. That is in the semi-oblivious chase only two triggers are applied: the active trigger $\tau_3$ and the representative of the equivalence class for $\tau_1$ and $\tau_2$. Hence, the resulted instance is $J^{\mathbf{sobl}} = I \cup \{T(d, X_2)\} \cup \{T(a, X_3)\}$. Finally, the oblivious chase will apply all triggers returning instance $J^{\mathbf{obl}}$. Bellow are the tabular representations of the instances resulted by applying the standard, semi-oblivious and oblivious-chase algorithms. The instances are restricted to relation $T$:

$J^{\mathbf{std}}$

| $T$ | |
|---|---|
| $a$ | $a$ |
| $d$ | $X_1$ |

$J^{\mathbf{sobl}}$

| $T$ | |
|---|---|
| $a$ | $a$ |
| $d$ | $X_2$ |
| $a$ | $X_3$ |

$J^{\mathbf{obl}}$

| $T$ | |
|---|---|
| $a$ | $a$ |
| $d$ | $X_4$ |
| $a$ | $X_5$ |
| $a$ | $X_6$ |

Similarly to the previous chase algorithms, we define termination classes over sets of tgds: $\mathsf{CT}^{\mathsf{sobl}}_{I,\exists}$,$\mathsf{CT}^{\mathsf{sobl}}_{I,\forall}$, $\mathsf{CT}^{\mathsf{sobl}}_{\forall\exists}$ and $\mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$ for the semi-oblivious algorithm. The following proposition shows that the nondeterministic behavior of the semi-oblivious-chase algorithm does not influence the termination for different execution branches.

▶ Proposition 21. Let $I$ be an instance. Then $\mathsf{CT}^{\mathsf{sobl}}_{I,\forall} = \mathsf{CT}^{\mathsf{sobl}}_{I,\exists}$ and $\mathsf{CT}^{\mathsf{sobl}}_{\forall\forall} = \mathsf{CT}^{\mathsf{sobl}}_{\forall\exists}$.

**Proof.** It follows from the observation that the set of representative trigger for each equivalence classes is the same for all execution branches. ◀

Similarly to the oblivious chase case, it can be shown that in case the semi-oblivious-chase algorithm terminates and not fails with the input instance $I$ and the set of tgds $\Sigma$, then the instances returned by each execution branch are isomorphically equivalent. In case we allow egds in $\Sigma$, then the returned instances will be homomorphically equivalent. In this case we denote by $chase^{\mathbf{sobl}}{}_{\Sigma}(I)$ one representative instance of the homomorphic equivalence class for the instances computed on each execution branch. In case the semi-oblivious-chase algorithm fails or it is non-terminating, we set $chase^{\mathbf{sobl}}{}_{\Sigma}(I) = \bot$.

The same as for the oblivious chase case, we can find a rewriting of the dependencies such that we can relate the termination of the semi-oblivious-chase algorithm to the termination of the standard-chase algorithm. We introduce a transformation, called *semi-enrichment*, that takes a tgd $\xi = \alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$ over a schema $\mathbf{R}$, and converts it into the tgd $\tilde{\xi} = \alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z}), H(\bar{x})$, where $H$ is a new relational symbol that does not appear in $\mathbf{R}$. For a set $\Sigma$ of tgds defined on schema $\mathbf{R}$, the transformed set is $\tilde{\Sigma} = \{\tilde{\xi} : \xi \in \Sigma\}$. Using the semi-enrichment notion, the relation between the standard and semi-oblivious-chase terminations can be presented as follows.

▶ **Theorem 22.** [26] *Let $\Sigma$ be a set of tgds and $I$ an instance. Then we have:*
1. $\Sigma \in \mathsf{CT}^{\mathsf{sobl}}_{I,\forall}$ *if and only if* $\tilde{\Sigma} \in \mathsf{CT}^{\mathsf{std}}_{I,\forall}$, *and*
2. $\Sigma \in \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$ *if and only if* $\tilde{\Sigma} \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.

The following theorem relates the instances returned by the semi-oblivious chase and the instances returned by the standard-chase algorithm.

▶ **Theorem 23.** [38] *Let $I$ be and instance and let $\Sigma$ be a set of tgds and egds, such that $chase^{\mathbf{sobl}}{}_{\Sigma}(I) \neq \bot$. Then $chase^{\mathbf{std}}{}_{\Sigma}(I) \leftrightarrow chase^{\mathbf{sobl}}{}_{\Sigma}(I)$ and $chase^{\mathbf{sobl}}{}_{\Sigma}(I) \models \Sigma$.*

Similarly to the standard chase algorithm, the problem of testing if the semi-oblivious chase is terminating for a given instance and a given set of tgds is undecidable [38]. The same result applies for the class $\mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$, if we allow at least one denial constraint [26]. It remains an open problem if the class $\mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$ remains undecidable when considering only sets of tgds.

The proposition below shows the set relationship between termination classes for the chase variations presented so far.

▶ Proposition 24. [26] $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$.

**Proof.** The inclusions are clear from the definition of the corresponding chase steps. For the first strict inclusion consider $\Sigma_1 = \{R(x, y) \to \exists z\ R(x, z)\}$. From Example 17, we know that $\Sigma_1 \notin \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$. On the other hand, it is easy to see that for any instance $I$ only a maximum of $|I|$ triggers will be applied on each execution branch by the semi-oblivious chase. For the second strict inclusion consider $\Sigma_2 = \{\xi_1, \xi_2\}$, where:

$$\begin{aligned}\xi_1 &= R(x) \to \exists z\ S(z), T(z, x), \text{ and}\\ \xi_2 &= S(x) \to \exists z\ R(z), T(x, z).\end{aligned}$$

It is easy to check that the standard chase terminates for $\Sigma_2$ with any input instance $I$. Additionally, the semi-oblivious chase does not terminate for $\Sigma_2$ and instance $I = \{R(a)\}$. ◀

### 3.3.3 The core chase

The class of chase algorithms is enriched by the core chase algorithm introduced by Deutsch et al. in [16]. We need to clarify from the very beginning that the core chase differs from the other variations by executing in parallel all applicable standard tgd chase steps and also computing the core of the unified instance. Note that we may only apply the standard tgd chase steps in parallel but not the egd chase steps, as the latter may modify the given instance by equating existing labeled nulls to constants or to other labeled nulls.

For a better understanding, we slightly changed the algorithm from [16] by applying all the egd triggers before applying in parallel all the active tgd triggers. This modification does not change however the result or the complexity of the given algorithm.

```
CORE-CHASE(I,$\Sigma$)
1  $I_0 := I$; $i := 0$;
2  if exists a standard egd trigger $(\xi, h)$ for $I_i$
3      then
4          if $I_i \xrightarrow{(\xi,h)} \perp$
5              then return FAIL
6              else  $I_i \xrightarrow{(\xi,h)} I_{i+1}$; $i := i + 1$ goto 2
7  if exists a standard tgd trigger for $I_i$
8      then
9          For all active trigger $(\xi, h)$ for $I_i$, compute in parallel $I_i \xrightarrow{\xi,h} J_j$
10         $I_{i+1} := Core(\bigcup_j J_j)$; $i := i + 1$
11     else
12         return $I_i$
13 goto 2
```

By applying all the triggers in parallel, the core-chase algorithm eliminates the non-deterministic part introduced by the standard-chase algorithm. In case the core-chase algorithm terminates in the finite and does not fail for input $I$ and $\Sigma$, we denote the returned instance by $chase^{\textbf{core}}{}_{\Sigma}(I)$ . In case the core chase fails or it is non-terminating, we set $chase^{\textbf{core}}{}_{\Sigma}(I) = \perp$.

Similarly to the other chase variations, for the core chase we introduce classes of tgd sets $\mathsf{CT}^{\mathsf{core}}_{I,\forall}$, $\mathsf{CT}^{\mathsf{core}}_{I,\exists}$, $\mathsf{CT}^{\mathsf{core}}_{\forall\forall}$ and $\mathsf{CT}^{\mathsf{core}}_{\forall\exists}$. Because the core chase is deterministic, it follows that $\mathsf{CT}^{\mathsf{core}}_{I,\forall} = \mathsf{CT}^{\mathsf{core}}_{I,\exists}$ for any instance $I$ and that $\mathsf{CT}^{\mathsf{core}}_{\forall\forall} = \mathsf{CT}^{\mathsf{core}}_{\forall\exists}$.

▶ **Theorem 25.** [16] *Let $I$ be an instance. Then* $\mathsf{CT}^{\mathsf{std}}_{I,\exists} \subset \mathsf{CT}^{\mathsf{core}}_{I,\forall}$ *and* $\mathsf{CT}^{\mathsf{std}}_{\forall\exists} \subset \mathsf{CT}^{\mathsf{core}}_{\forall\forall}$.

**Proof.** For the second strict inclusion consider $\Sigma = \{R(x) \to \exists y\ R(y), S(x)\}$. Clearly the standard chase does not terminate on any branch with $I = \{R(a)\}$ and $\Sigma$, that is $\Sigma \notin \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$. On the other hand, for any instance $I$, the core chase will terminate in maximum $|I_R|$ steps, where $I_R$ is instance $I$ restricted to tuple over relation $R$. ◀

In [16] it is shown that the membership problem for the class $\mathsf{CT}^{\mathsf{core}}_{I,\forall}$ for a given instance $I$ is RE-complete. To this, it was shown most recently [26] that the membership problem for the class $\mathsf{CT}^{\mathsf{core}}_{\forall\forall}$ is coRE-complete.

It may be noted that at line 10 the core-chase algorithm does not simply compute the union between all the instances computed at line 9, but it also computes its core. This gives the following link between the core chase and standard-chase algorithms:

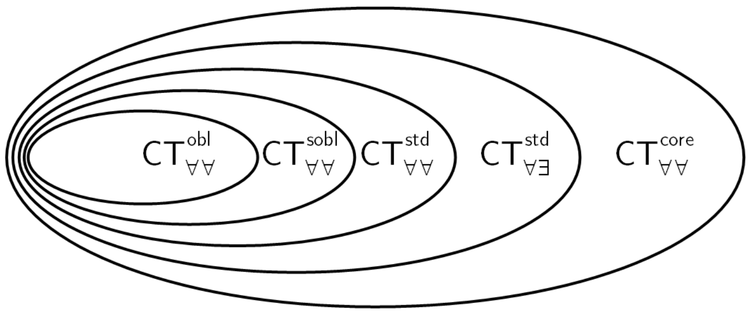
$\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$
$\mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$
$\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$
$\mathsf{CT}^{\mathsf{std}}_{\forall\exists}$
$\mathsf{CT}^{\mathsf{core}}_{\forall\forall}$

**Figure 1** Termination classes for different chase variations.

▶ **Theorem 26.** *Let $I$ be and instance and let $\Sigma$ be a set of tgds and egds, such that $chase^{\mathbf{std}}{}_{\Sigma}(I) \neq \perp$. Then $chase^{\mathbf{std}}{}_{\Sigma}(I) \leftrightarrow chase^{\mathbf{core}}{}_{\Sigma}(I)$ and $chase^{\mathbf{core}}{}_{\Sigma}(I) \models \Sigma$, even more, $Core(chase^{\mathbf{std}}{}_{\Sigma}(I)) = chase^{\mathbf{core}}{}_{\Sigma}(I)$.*

Before ending this section about chase variations, let us summarize the differences between the presented chase algorithms. First we saw that in case the algorithm terminates and does not fail with input $I$ and $\Sigma$, then the instances returned by all of the presented chase variations are homomorphically equivalent. We also saw that the complexity of testing the existence of a trigger is slightly easier than testing the existence of an active trigger. From this it follows that the oblivious and semi-oblivious-chase steps are less expensive than the standard-chase step that is also less expensive than the core-chase step. On the other hand, a set of dependencies is more likely to terminate for the core chase than any of the other chase variations. Figure 1 shows the set inclusion relationship between different termination classes.

## 4 Sufficient conditions for the chase termination

In the previous section we saw that it is undecidable to test for all the chase variations if the chase will terminate for a given instance and a given set of dependencies. This motivated the research community to find large classes of tgd sets that ensure termination of the standard-chase algorithm for all instances. In this section some of these classes of tgd sets will be presented for which it is known that the standard chase terminates on all execution branches for all input instances. We also investigate here if these classes are sufficient to guarantee the chase termination for other chase variations beside the standard chase. For this, we will say that a class of sets of tgds $\mathcal{C}$ is closed under enrichment if $\Sigma \in \mathcal{C}$ implies $\hat{\Sigma} \in \mathcal{C}$. Similarly, the class $\mathcal{C}$ is closed under semi-enrichment if $\Sigma \in \mathcal{C}$ implies $\tilde{\Sigma} \in \mathcal{C}$. From Theorems 18 and 22 it directly follows:

▶ **Corollary 27.** *Let $\mathcal{C} \subseteq \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.*

1. *if $\mathcal{C}$ is closed under enrichment, then $\mathcal{C} \subseteq \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$.*
2. *if $\mathcal{C}$ is closed under semi-enrichment, then $\mathcal{C} \subseteq \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$.*

We will use the previous corollary to show that the termination classes presented here not only guarantee the termination for the standard-chase algorithm but also for the semi-oblivious-chase algorithm.

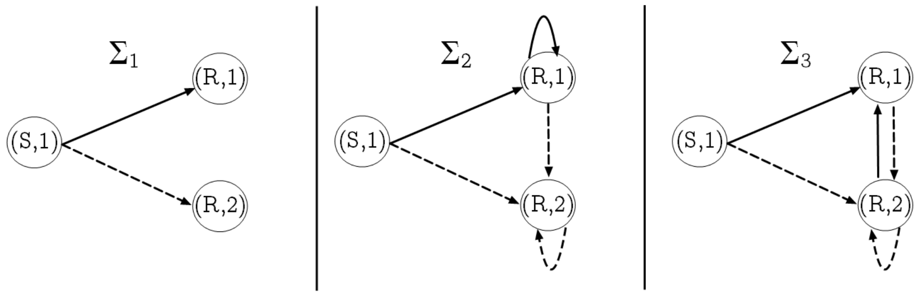

**Figure 2** Extended dependency graphs associated with dependencies from Example 30.

## 4.1 Rich acyclicity

The class of *richly acyclic* set of dependencies was introduced by Hernich and Schweikardt in [30] in a different context, and it was shown in [25] that this class guarantees termination for the oblivious chase on any input instance.

▶ **Definition 28.** [19, 12] For a given database schema $\mathbf{R}$ define a *position* in $\mathbf{R}$ to be a pair $(R, k)$, where $R$ is a relation symbol in $\mathbf{R}$ and $k$ a natural number with $1 \leq k \leq arity(R)$, such that $k$ identifies the $k$-th element in $R$.

The notion of extended-dependency graph is defined as follows:

▶ **Definition 29.** [30] Let $\Sigma$ be a set of tgds over schema $\mathbf{R}$. The *extended-dependency graph* associated with $\Sigma$ is a directed edge-labeled graph $G^E_\Sigma = (V, E)$, such that each vertex represents a position in $\mathbf{R}$ and $((R, i), (S, j)) \in E$, if there exists a tgd $\xi \in \Sigma$ of the form $\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$, and if one of the following holds:

1. $x \in \bar{x}$ and $x$ occurs in $\alpha$ on position $(R, i)$ and in $\beta$ on position $(S, j)$. In this case the edge is labeled as universal;
2. $x \in \bar{x} \cup \bar{y}$ and $x$ occurs in $\alpha$ on position $(R, i)$ and variable $z \in \bar{z}$ that occurs in $\beta$ on position $(S, j)$. In this case the edge is labeled as existential.

The following example illustrates the previous definitions:

▶ **Example 30.** Consider database schema $\mathbf{R} = \{S, R\}$, with $arity(S) = 1$ and $arity(R) = 2$. The set $\{(S, 1), (R, 1), (R, 2)\}$ represents all positions in $\mathbf{R}$. Let $\Sigma_1$ contain the following dependency over $\mathbf{R}$:

$$\xi_{11} \quad = \quad S(x) \to \exists y\ R(x, y)$$

let $\Sigma_2$ contain the following dependencies:

$$\begin{aligned} \xi_{21} &= S(x) \to \exists y\ R(x, y), \text{ and} \\ \xi_{22} &= R(x, y) \to \exists z\ R(x, z) \end{aligned}$$

and, finally, let $\Sigma_3$ be a slight modification of $\Sigma_2$:

$$\begin{aligned} \xi_{31} &= S(x) \to \exists y\ R(x, y), \text{ and} \\ \xi_{32} &= R(x, y) \to \exists z\ R(y, z). \end{aligned}$$

Figure 2 captures the extended dependency graphs associated with $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$ (note that the existential edges are represented as dotted lines).

▶ **Definition 31.** [30] A set of tgds $\Sigma$ is said to be *richly acyclic* if its extended dependency graph does not contain a cycle going through an existential edge. We denote by $\mathsf{RA}$ the class of all richly acyclic tgd sets.

Note that the problem of testing if $\Sigma \in \mathsf{RA}$ is polynomial in size of $\Sigma$. Returning to Example 30, $\Sigma_1$ is richly acyclic because it does not contain any cycles. On the other hand, neither $\Sigma_2$ or $\Sigma_3$ are richly acyclic. As we will see in the following subsection, the $\mathsf{RA}$ ensures termination for the standard-chase algorithm on any input instance, that is $\mathsf{RA} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. The next theorem follows directly from Corollary 27 and the observation that $\mathsf{RA}$ is closed under enrichment.

▶ **Theorem 32.** [25] *Let $\Sigma \in \mathsf{RA}$ and let $I$ be an instance. Then there exists a polynomial in size of $I$ that bounds the length of every oblivious-chase sequence of $I$ and $\Sigma$.*

Mainly, the previous result states that $\mathsf{RA} \subseteq \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ and based on the termination hierarchy represented in Figure 1, it follows that any set of tgds from $\mathsf{RA}$ ensures termination of any of the chase variation previously presented on any input instances.

## 4.2 Weak acyclicity

Fagin et al. [19] introduced the class of *weakly acyclic* dependencies as a class of sets of tgds that ensures standard-chase termination on all execution branches for all input instances. Intuitively weak acyclicity checks if the set of tgds does not have a cyclic condition such that another new null value forces the adding of a new null value.

▶ **Definition 33.** [19] Let $\Sigma$ be a set of tgds over schema $\mathbf{R}$. The *dependency graph* associated with $\Sigma$ is a directed edge-labeled graph $G_\Sigma = (V, E)$, such that the set of vertexes $V$ represents the positions in $\mathbf{R}$. There is an edge $((R,i),(S,j)) \in E$, if there exists a dependency $\xi \in \Sigma$ of the form $\alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z})$. There exists $x \in \bar{x}$ such that $x$ occurs in position $(R,i)$ in $\alpha$ and if one of the following holds:
1. $x$ occurs in $\beta$ in position $(S,j)$. In this case the edge is labeled as universal;
2. there exists variable $z \in \bar{z}$ which occurs in position $(S,j)$ in $\beta$. In this case the edge is labeled as existential.

▶ **Definition 34.** [19] A set of tgds $\Sigma$ is said to be *weakly acyclic* if the corresponding dependency graph does not have any cycle going through an existential edge. By $\mathsf{WA}$ is denoted the class of all weakly acyclic sets of tgds.

Note that the problem of testing if $\Sigma \in \mathsf{WA}$ is polynomial in size of $\Sigma$. Figure 3 illustrates the dependency graphs associated with the dependencies from Example 30. Based on the previous definition, it follows that : $\Sigma_1$ is weakly acyclic as the dependency graph does not contain any cycles; $\Sigma_2$ is weakly acyclic as its dependency graph has a cycle going only through universal edges; $\Sigma_3$ is not weakly acyclic as it has a cycle going through an existential edge. From the definitions of the $\mathsf{RA}$ and $\mathsf{WA}$ classes, it follows that $\mathsf{RA} \subseteq \mathsf{WA}$. Also because $\Sigma_2 \in \mathsf{WA}$ and $\Sigma_2 \notin \mathsf{RA}$, it follows that the inclusion is strict, that is $\mathsf{RA} \subset \mathsf{WA}$.

▶ **Theorem 35.** [19] *Let $\Sigma \in \mathsf{WA}$ and let $I$ be an instance. Then there exists a polynomial in size of $I$ that bounds the length of every standard-chase sequence of $I$ and $\Sigma$.*

From the chase termination hierarchy it follows that if $\Sigma \in \mathsf{WA}$, then $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$ and $\Sigma \in \mathsf{CT}^{\mathsf{core}}_{\forall\forall}$. Besides, even if $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ the classes $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ and $\mathsf{WA}$ are incomparable. For this consider the tgd set $\Sigma$ from Example 17. It is easy to see that $\Sigma \in (\mathsf{WA} \setminus \mathsf{CT}^{\mathsf{obl}}_{\forall\forall})$. For the other direction consider $\Sigma' = \{S(y), R(x,y) \to \exists z\ R(y,z)\}$, clearly $\Sigma' \in (\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} \setminus \mathsf{WA})$.

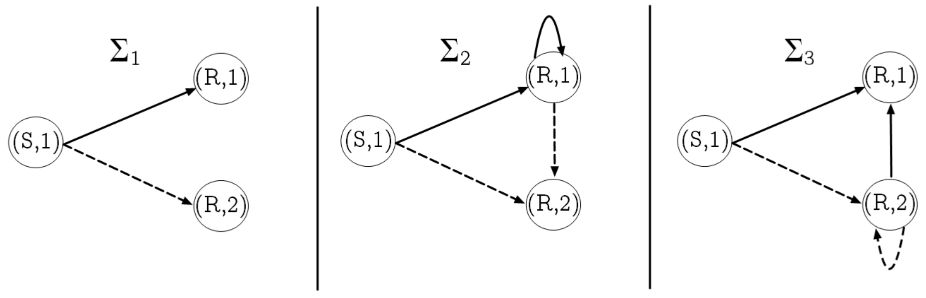

**Figure 3** Dependency graphs associated with dependencies from Example 30.

From the observation that the class WA is closed under semi-enrichment, Theorem 35 and Corollary 27, it follows:

▶ **Theorem 36.** $\mathsf{WA} \subset \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$.

**Proof.** For the strict inclusion part of this theorem consider the same set of dependencies $\Sigma' = \{S(y), R(x,y) \to \exists z\ R(y,z)\}$. ◀

## 4.3 Safe dependencies

Meier, Schmidt and Lausen [41] observed that the weak acyclicity condition takes into account nulls that may not create infinite standard-chase sequences. For example, consider the set $\Sigma = \{\xi\}$ [41] where:

$$\xi \;=\; R(x,y,z), S(y) \to \exists w\ R(y,w,x).$$

Figure 4a) represents the corresponding dependency graph with cycles going through existential edges involving position $(R,2)$. Thus, the dependency is not weakly acyclic. On the other hand, the newly created null in position $(R,2)$ may create new null values only if the same null also appears in position $(S,1)$. Based on the given dependency, new nulls cannot be generated in position the $(S,1)$. Hence, this dependency can not cyclically create new nulls.

In order to introduce the notion of safe dependencies, we first need to define the following concept.

▶ **Definition 37.** [12] The *affected positions* associated with a set of tgds $\Sigma$ is the set $\mathit{aff}(\Sigma)$ defined as follows. For all positions $(R,i)$ that occur in the head of some tgd $\xi \in \Sigma$, then

1. if an existential variable appears in position $(R,i)$ in $\xi$, then $(R,i) \in \mathit{aff}(\Sigma)$;
2. if universally quantified variable $x$ appears in position $(R,i)$ in the head and $x$ appears only in affected positions in the body, then $(R,i) \in \mathit{aff}(\Sigma)$.

Intuitively, the affected positions are those where new null values can occur during the chase process. For example, the set of affected positions associated with the set of dependencies $\Sigma = \{R(x,y,z), S(y) \to \exists w\ R(y,w,x)\}$ is $\mathit{aff}(\Sigma) = \{(R,2)\}$.

▶ **Definition 38.** [41] The *propagation graph* for a set of tgds $\Sigma$ is a directed edge labeled graph $P_\Sigma = (\mathit{aff}(\Sigma), E)$. Where $((R,i),(S,j)) \in E$ if there exists a dependency $\xi \in \Sigma$ of the form $\alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z})$, there exists a variable $x$ that occurs in $\alpha$ in position $(R,i)$, $x$ occurs only in affected positions in $\alpha$ and one of the following holds:

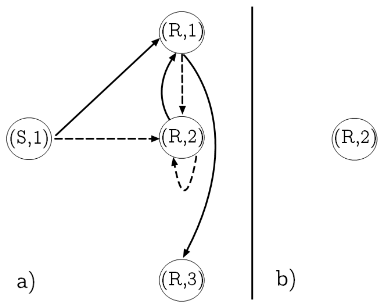

**Figure 4** a) Dependency graph, b) Propagation graph for $\{R(x, y, z), S(y) \to \exists w\ R(y, w, x)\}$.

1. $x$ appears in $\beta$ in affected position $(S, j)$. In this case the edge is labeled as universal;
2. there exists variable $z \in \bar{z}$ which occurs in position $(S, j)$ in $\beta$. In this case the edge is labeled existential.

Considering the same dependency set $\Sigma = \{R(x, y, z), S(y) \to \exists w\ R(y, w, x)\}$, Figure 4a) represents the corresponding dependency graph and Figure 4b) the corresponding propagation graph. Note that the propagation graph contains only one node, corresponding to the affected position $(R, 2)$. Because $y$ appears in the head in a non-affected position $(S, 1)$, it follows that there are no edges in the propagation graph.

▶ **Definition 39.** [41] A set of tgds $\Sigma$ is called *safe* if its propagation graph $P_\Sigma$ does not have a cycle going through an existential edge. By SD is denoted the class of all safe sets of tgds.

Note that the problem of testing if $\Sigma \in \mathsf{SD}$ is polynomial in size of $\Sigma$. In our example, the dependency graph did not contain any edges (see Figure 4b)), hence $\Sigma \in \mathsf{SD}$.

▶ **Theorem 40.** [41] *Let $\Sigma \in \mathsf{SD}$. Then there exists a polynomial in size of $I$ that bounds the length of every standard-chase sequence of $I$ and $\Sigma$.*

From our running example in this subsection and from the definition of the SD class, it follows that $\mathsf{WA} \subset \mathsf{SD}$. Also, similarly to the weakly acyclic class it can be shown that SD is closed under semi-enrichment. Thus, because the previous theorem states that $\mathsf{SD} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, it follows that actually $\mathsf{SD} \subset \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. This means that given a set of tgds which is safe, one may use the semi-oblivious chase to compute an instance which is (see previous section) homomorphically equivalent to any instance returned by the standard chase with the same input. Finally, it can be easily noted that, even if SD is bigger than WA, it does not contain the termination class $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$. For this consider $\Sigma = \{R(x, x) \to \exists z\ R(x, y)\}$, clearly $\Sigma \in (\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} \setminus \mathsf{SD})$.

## 4.4 Super weak acyclicity

The following class of sets of tgds properly extends the class of safe sets of tgds and consequently the class of weakly acyclic and richly acyclic sets of tgds. The new class of dependencies, introduced by Marnette [38], beside omitting the nulls that can't generate infinite chase sequences, as in the case of safe dependencies, also takes into account the repeating variables. For a more uniform presentation of the sufficient classes, we will slightly change the notations used in [38].

In this subsection we assume that the set of dependencies $\Sigma$ has distinct variable names in each tgd. We also assume that there exists a total order between the atoms in each dependency. With this, we can now define the *atom position* to be a triple $(\xi, R, i)$, where $\xi$ is a dependency in $\Sigma$, $R$ is a relation name that occurs in $\xi$, $i$ a positive integer $i \leq n$, where $n$ is the maximum number of occurrences of $R$ in $\xi$, given by the total order between the atoms in the tgd. Clearly each atom position uniquely identifies an atom in $\Sigma$. Similarly to the notion of position, a *place* can be defined to be a pair $((\xi, R, i), k)$, where $(\xi, R, i)$ is an atom position and $1 \leq k \leq arity(R)$. Intuitively, the place identifies the variable that appears in the $k$-th attribute in the atom represented by $(\xi, R, i)$.

Let $Var(\xi)$ denote the set of variables that occurs in dependency $\xi$. As mentioned, for any two distinct dependencies $\xi_1$ and $\xi_2$ from $\Sigma$, we have $Var(\xi_1) \cap Var(\xi_2) = \emptyset$. The mapping $Var$ is extended in the natural way to a set of dependencies $\Sigma$, $Var(\Sigma) = \cup_{\xi \in \Sigma} Var(\xi)$. Similarly, we define the mappings $Var^{\exists}$ and $Var^{\forall}$ that map each dependency $\xi$ to the set of existentially quantified variables in $\xi$ and to the universally quantified variables in $\xi$, respectively. Clearly for each dependency $\xi$, $Var^{\exists}(\xi)$ and $Var^{\forall}(\xi)$ represent a partition of $Var(\xi)$.

Given a tgd $\xi$ and $y \in Var^{\exists}(\xi)$, $\mathsf{Out}(\xi, y)$ is defined to be the set of places in the head of $\xi$ where $y$ occurs. Given a set a tgd $\xi$ and $x \in Var^{\forall}(\xi)$, $\mathsf{In}(\xi, x)$ is defined to be the set of places in the body of $\xi$ where $x$ occurs. Intuitively, $\mathsf{Out}(\xi, y)$ represents the places where variable $y$ is "exported" when applying tgd $\xi$. Similarly, $\mathsf{In}(\xi, x)$ represents the places that need to be "filled" for variable $x$ in order for $\xi$ to be applied.

Given an atom position $(\xi, R, i)$, a substitution $\theta$ is a function that maps each variable $x$ that occurs in the atom $(\xi, R, i)$ to a constant if $x \in Var^{\forall}(\xi)$ and to a fresh new constant, if $x \in Var^{\exists}(\xi)$, where by "new fresh" we mean the next unused element in some fixed enumeration of the constants. The atom resulted by replacing each variable in the atom given by $(\xi, R, i)$ with the substitution $\theta$ is denoted by $\theta(\xi, R, i)$. Two atoms $(\xi_1, R, i_1)$ and $(\xi_2, R, i_2)$ are said to be *unifiable* if there exist substitutions $\theta_1$ and $\theta_2$ such that $\theta_1(\xi_1, R, i_1) = \theta_2(\xi_2, R, i_2)$. Two places $p_1 = ((\xi_1, R, i_1), k_1)$ and $p_2 = ((\xi_2, R, i_2), k_2)$ are said to be *unifiable* if $k_1 = k_2$ and $(\xi_1, R, i_1)$ is unifiable with $(\xi_2, R, i_2)$. By $p_1 \sim p_2$ it is denoted that $p_1$ and $p_2$ are unifiable. Let us define $\Gamma_\Sigma$ to be a function that maps each variable $x$ to the set of places where $x$ occurs in $\Sigma$. $\Gamma_\Sigma^H$ represents the function that maps each variable $x$ to the set of places from the head of some dependency where $x$ occurs. Similarly, the function $\Gamma_\Sigma^B$ maps each variable $x$ to the set of places from the body of some dependency where $x$ occurs.

For a better understanding of the previous notions, let us consider the example:

▶ **Example 41.** [47] Let $\Sigma = \{\xi_1, \xi_2\}$, where:

$$\begin{aligned} \xi_1 &= R(x) \to \exists y, z\ S(x, y, z), \text{ and} \\ \xi_2 &= S(v, w, w) \to R(w). \end{aligned}$$

The atom positions for $\Sigma$ are: $(\xi_1, R, 1)$, corresponding with the atom $R(x)$; $(\xi_1, S, 1)$, corresponding with the atom $S(x, y, z)$; $(\xi_2, S, 1)$, corresponding with the atom $S(v, w, w)$; and $(\xi_2, R, 1)$, corresponding with the atom $R(w)$. We have $Var^{\forall}(\Sigma) = \{x, v, w\}$ and $Var^{\exists}(\Sigma) = \{y, z\}$. Clearly $(\xi_1, R, 1)$ is unifiable with $(\xi_2, R, 1)$, consider for example unifiers $\theta_1 = \{x/a\}$ and $\theta_2 = \{w/a\}$ where both variables $x$ and $w$ are in $Var^{\forall}(\Sigma)$. On the other hand, the atom positions $(\xi_1, S, 1)$ and $(\xi_2, S, 1)$ are not unifiable because $y$ and $z$ are existentially quantified variables and thus any unifier will map $y$ and $z$ to distinct constants (recall that each existential variable is mapped to a new fresh constants). Hence, we only have

$((\xi_1, R, 1), 1) \sim ((\xi_2, R, 1), 1)$. For variable $x$ the set $\Gamma_\Sigma(x) = \{((\xi_1, R, 1), 1), ((\xi_1, S, 1), 1)\}$, $\Gamma^B_\Sigma(x) = \{((\xi_1, R, 1), 1)\}$ and $\Gamma^H_\Sigma(x) = \{((\xi_1, S, 1), 1)\}$.

Given two sets of places $P$ and $Q$, we denote $P \sqsubseteq Q$ if for all $p \in P$ there exists $q \in Q$ such $p \sim q$. Let us now define mapping $\mathsf{Move}(\Sigma, Q)$ that gives the smallest set of places $P$ such that $Q \subseteq P$, and for all variables $x$ that occurs in a body of some dependency $\xi \in \Sigma$ if $\Gamma^B_\Sigma(x) \sqsubseteq P$ then $\Gamma^H_\Sigma(x) \subseteq P$. Intuitively, the $\mathsf{Move}(\Sigma, Q)$ returns the smallest set of places such that new atoms may be generated in those positions by chasing some atoms given by the places in $Q$.

▶ **Definition 42.** [38] Given $\Sigma$ a set of tgds and $\xi_1, \xi_2 \in \Sigma$, we say $\xi_1$ triggers $\xi_2$ in $\Sigma$, and it is denoted by $\xi_1 \rightsquigarrow_\Sigma \xi_2$, iff there exist a variable $y \in Var^\exists(\xi_1)$ and a variable $x \in Var^\forall(\xi_2)$ occurring in both the body and the head of $\xi_2$ such that:

$$\mathsf{In}(\xi_2, x) \sqsubseteq \mathsf{Move}(\Sigma, \mathsf{Out}(\xi_1, y)).$$

▶ **Definition 43.** [38] A set of tgds $\Sigma$ is said to be *super-weakly acyclic* iff the trigger relation $\rightsquigarrow_\Sigma$ is acyclic. We denote by $\mathsf{SwA}$ the set off all super-weakly acyclic tgd sets.

▶ **Example 44.** Let us consider the same set of dependencies $\Sigma = \{\xi_1, \xi_2\}$ from Example 41. The place $((\xi_1, S, 1), 1)$ is not unifiable with $((\xi_2, S, 1), 1)$, thus $\mathsf{In}(\xi_2, \mathsf{w}) \not\sqsubseteq \mathsf{Move}(\Sigma, \mathsf{Out}(\xi_1, y))$, that is $\xi_1 \not\rightsquigarrow_\Sigma \xi_2$. Similarly, $\xi_2$ does not contain any existential variables and so it follows that $\xi_2 \not\rightsquigarrow_\Sigma \xi_1$. As both dependencies do not share common relation names in the head and body, it follows that $\xi_1 \not\rightsquigarrow_\Sigma \xi_1$ and $\xi_2 \not\rightsquigarrow_\Sigma \xi_2$. That is the relation $\rightsquigarrow_\Sigma$ does not induce any cycle, following that $\Sigma$ is super-weakly acyclic. Moreover, it can be seen that $\Sigma$ is not safe as between the affected positions $(R, 1)$ and $(S, 2)$ there exists a cycle through an existential edge in the corresponding propagation graph.

Marnette [38] showed that the membership problem *Is* $\Sigma \in \mathsf{SwA}$*?* is polynomial in the size of $\Sigma$. Spezzano and Greco [47] also proved that $\mathsf{SD} \subset \mathsf{SwA}$, that is the super-weak acyclic class properly contains the safe dependencies. The class $\mathsf{SwA}$ is closed in adding atoms to the body of dependencies. Thus given a set of tgds $\Sigma$, then any set of dependencies $\Sigma'$ obtained from $\Sigma$ by adding new atoms in the body of any dependency remains super-weakly acyclic.

▶ **Theorem 45.** [38] *Let* $\Sigma \in \mathsf{SwA}$ *and let* $I$ *be an instance. Then there exists a polynomial in the size of* $I$ *that bounds the length of every semi-oblivious-chase sequence of* $I$ *and* $\Sigma$*. Thus,* $\mathsf{Swa} \subset \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$.

This concludes that the super-weakly acyclic class of tgd sets is sufficient for the termination of the semi-oblivious, standard and core-chase algorithms for all input instances.

## 4.5 Stratification

The stratified model of dependencies was introduced by Deutsch et al. in [16]. This class relaxes the condition imposed by weak acyclicity by stratifying the dependencies and check for weak acyclicity on each of these strata instead of checking for the entire set.

▶ **Definition 46.** [16] Let $\xi_1$ and $\xi_2$ be two tgds , we write $\xi_1 \prec \xi_2$, if there exist the instances $I$, $J$ and a vector $\bar{a} \subseteq dom(J)$ such that:

1. $I \models \xi_2(\bar{a})$, and
2. there exists an active trigger $(\xi_1, h)$, such that $I \xrightarrow{(\xi_1, h)} J$, and
3. $J \not\models \xi_2(\bar{a})$.

▶ **Example 47.** Consider $\Sigma = \{\xi_1, \xi_2\}$, where:

$$\begin{aligned} \xi_1 &= R(x,y) \to S(x), \text{ and} \\ \xi_2 &= S(x) \to R(x,x). \end{aligned}$$

With the instance $I = \{R(a,b)\}$ and the vector $\bar{a} = (a)$ we have that $I \models \xi_2(a)$; and for the homomorphism $h = \{x/a, y/b\}$ we have $I \xrightarrow{(\xi_1,h)} J$, where $J = \{R(a,b), S(a)\}$. Because $J \not\models \xi_2(a)$, it follows that $\xi_1 \prec \xi_2$. On the other hand, $\xi_2 \not\prec \xi_1$ because for any instance $I$ and vector of constants $\bar{b}$ such that $I \models \xi_1(\bar{b})$ and $I \xrightarrow{(\xi_2,h)} J$, for some active trigger $(\xi_2, h)$, it follows that $J \models \xi_2(\bar{b})$.

The authors of [16] claimed that testing if $\xi_1 \prec \xi_2$ is in NP, as we will show next, this cannot be true unless NP = coNP.

▶ **Theorem 48.** [26] *Given two tgds $\xi_1$ and $\xi_2$, the problem of testing if $\xi_1 \prec \xi_2$ is* coNP*-hard.*

**Proof.** We will use a reduction from the graph 3-colorability problem that is known to be NP-complete. It is also known that a graph $G$ is 3-colorable if and only if there exists a homomorphism from $G$ to $K_3$, where $K_3$ is the complete graph with 3 vertices.

A graph $G = (V,E)$, where $V = n$ and $E = m$, is identified with the sequence $G(x_1, \ldots, x_n) = E(x_{i_1}, y_{i_1}), \ldots, E(x_{i_m}, y_{i_m})$ and treat the elements in $V$ as variables. Similarly, we identify the graph $K_3$ with the sequence

$$K_3(z_1,z_2,z_3) = E(z_1,z_2), E(z_2,z_1), E(z_1,z_3), E(z_3,z_1), E(z_2,z_3), E(z_3,z_2)$$

where $z_1, z_2$, and $z_3$ are variables. With these notations, given a graph $G = (V,E)$, we construct tgd's $\xi_1$ and $\xi_2$ as follows:

$$\begin{aligned} \xi_1 &= R(z) \to \exists z_1, z_2, z_3\ K_3(z_1,z_2,z_3), \text{ and} \\ \xi_2 &= E(x,y) \to \exists x_1, \ldots, x_n\ G(x_1, \ldots, x_n). \end{aligned}$$

Clearly the reduction is polynomial in the size of $G$. We will now show that $\xi_1 \prec \xi_2$ iff $G$ is not 3-colorable.

First, suppose that $\xi_1 \prec \xi_2$. Then there exists an instance $I$ and homomorphisms $h_1$ and $h_2$, such that $I \models h_2(\xi_2)$. Consider $J$, where $I \xrightarrow{(\xi_1,h_1)} J$. Thus $R^I$ had to contain at least one tuple, and $E^I$ had to be empty, because otherwise the monotonicity property of the chase we would imply that $J \models h_2(\xi_2)$.

On the other hand, we have $I \xrightarrow{(\xi_1,h_1)} J$, where $J = I \cup \{K_3(h_1'(z_1), h_1'(z_2), h_1'(z_3))\}$, and $h_1'$ is a distinct extension of $h_1$. Since $E^I = \emptyset$, and we assumed that $J \not\models h_2(\xi_2)$, it follows that there is no homomorphism from $G$ into $J$, i.e. there is no homomorphism from $G(h_2'(x_1), \ldots, h_2'(x_n))$ to $K_3(h_1'(z_1), h_1'(z_2), h_1'(z_3))$, where $h_2'$ is a distinct extension of $h_2$. Therefore the graph $G$ is not 3-colorable.

For the other direction, let us suppose that graph $G$ is not 3-colorable. This means that clearly there is no homomorphism from $G$ into $K_3$. In fact, with these assumptions let us consider $I = \{R(a)\}$, and the two homomorphisms $h_1 = \{z/a\}$ and $h_2 = \{x/h_1'(z_1), y/h_1'(z_2)\}$. It is easy to verify that $I$, $h_1$ and $h_2$ satisfy the three conditions for $\xi_1 \prec \xi_2$. ◀

The obvious upper bound for the problem $\xi_1 \prec \xi_2$ is $\Sigma_2^P$. In [26] it is shown that this upper bound can be lowered to $\Delta_2^P$. To the best of our knowledge these are the tidiest bounds found so far for the given problem.

Given a set of tgds $\Sigma$, the *chase graph* associated with $\Sigma$ is a directed graph $G = (V,E)$, where $V = \Sigma$ and $(\xi_1, \xi_2) \in E$ iff $\xi_1 \prec \xi_2$.

▶ **Definition 49.** [16] A set of tgds $\Sigma$ is said to be *stratified* if the set of dependencies in every simple cycle in the corresponding chase graph is weakly acyclic. The set of all stratified tgd sets is denoted by $\mathsf{Str}$.

In [26] it is shown that the complexity of testing if $\Sigma \in \mathsf{Str}$, for a given $\Sigma$, is in $\Pi_2^P$. Meier et al. [41] shown that $\mathsf{Str} \not\subseteq \mathsf{CT}_{\forall\forall}^{\mathsf{std}}$ but actually $\mathsf{Str} \subseteq \mathsf{CT}_{\forall\exists}^{\mathsf{std}}$, that is the stratification guarantees the termination only on some standard-chase execution branches for all input instances.

▶ **Example 50.** [41] Consider $\Sigma = \{\xi_1, \xi_2, \xi_3, \xi_4\}$, where:

$$\begin{aligned}
\xi_1 &= R(x) \to S(x,x); \\
\xi_2 &= S(x,y) \to \exists z\, T(y,z); \\
\xi_3 &= S(x,y) \to T(x,y), T(y,z); \text{ and} \\
\xi_4 &= T(x,y), T(x,z), T(z,x) \to R(y).
\end{aligned}$$

We have $\Sigma \in \mathsf{Str}$ is stratified since $\xi_1 \prec \xi_2$, $\xi_1 \prec \xi_3 \prec \xi_4 \prec \xi_1$, and the set $\{\xi_1, \xi_3, \xi_4\}$ is weakly acyclic. Let $I = \{R(a)\}$. The standard-chase execution branch that triggers repeatedly dependencies $\xi_1$, $\xi_2$, $\xi_3$ and $\xi_4$ never terminates. On the other hand, the chase sequences that never trigger $\xi_2$ will terminate.

Meier et al. [41] changed the stratification definition in order to guarantee termination on all execution branches for all instances.

▶ **Definition 51.** [41] Let $\xi_1$ and $\xi_2$ be two tgds we write $\xi_1 \prec_{\mathsf{C}} \xi_2$, if there exist instances $I$, $J$ and tuple $\bar{a} \subseteq dom(J)$, such that:

1. $I \models \xi_2(\bar{a})$, and
2. there exists trigger $(\xi_1, h)$, such that $I \xrightarrow{*,(\xi_1,h)} J$, and
3. $J \not\models \xi_2(\bar{a})$.

Given a set of tgds $\Sigma$, the *c-chase graph* associated with $\Sigma$ is a directed graph $G_{\mathsf{C}} = (V, E)$. With $V = \Sigma$ and $(\xi_1, \xi_2) \in E$ iff $\xi_1 \prec_{\mathsf{C}} \xi_2$. A set of tgds $\Sigma$ is said to be *c-stratified* if the set of dependencies in every simple cycle in the c-chase graph is weakly acyclic. The set of all c-stratified tgd sets is denoted by $\mathsf{CStr}$.

▶ **Theorem 52.** [41] *Let $\Sigma \in \mathsf{CStr}$ and let $I$ be an instance. Then there exists a polynomial, in size of $I$, that bounds the length of every standard chase sequence of $I$ and $\Sigma$.*

Using the same reduction as in Theorem 48, it can be shown that the problem of testing if $\xi_1 \prec_{\mathsf{C}} \xi_2$ is coNP-hard and it is in $\Delta_2^P$. Also the problem of testing if $\Sigma \in \mathsf{CStr}$, for a given $\Sigma$, it is in $\Pi_2^P$.

Meier et al. [41] showed that $\mathsf{WA} \subset \mathsf{CStr}$ and that $\mathsf{SD} \nparallel \mathsf{CStr}$[1]. Also Spezzano and Greco [47] proved that $\mathsf{SwA} \nparallel \mathsf{CStr}$, that is the super-weakly acyclic class is not comparable with the c-stratified class. Also based on the observation that $\mathsf{CStr}$ is closed under semi-enrichment, it follows that $\mathsf{CStr} \subset \mathsf{CT}_{\forall\forall}^{\mathsf{sobl}}$.

[1] The notation $A \nparallel B$ is shorthand for $A \not\subseteq B$ and $A \not\supseteq B$

## 4.6 Inductively restricted dependencies

Another class of dependencies that guarantees the standard-chase termination is the inductively restricted set of tgds. Note that the stratification method lifts the weakly acyclic class of dependencies to the class of c-stratified dependencies. The inductively restricted class generalizes the stratification method while still keeping the termination property for the standard-chase algorithm. This generalization is done using the so-called *restriction systems* [41]. With the help of the restriction systems, Meier et al. [41] define the new sufficient condition called *inductive restriction* that guarantees the standard-chase algorithm termination on all execution branches for all instances. From this condition a new hierarchy of classes of dependencies is revealed with the same termination property, called the *T-hierarchy*. Note that the inductive restriction condition presented here is given from the erratum (http://arxiv.org/abs/0906.4228) and not from [41], where the presented condition, as mentioned in the erratum, does not guarantee the standard-chase termination on all branches for all instances.

Let $\Sigma$ be a set of tgds, $I$ an instance and $A$ a set of nulls. The set of all positions $(R, i)$ such that there exists a tuple in $I$ that contains a variable from $A$ in position $(R, i)$ is denoted by null-pos$(A, I)$.

Similarly to relation "$\prec$" for the stratified dependencies, the binary relation "$\prec_P$" is defined for the inductive restriction condition, where $P$ is a set of positions.

▶ **Definition 53.** [41] Let $\Sigma$ be a set of tgds and $P$ a set of positions. Let $\xi_1$, $\xi_2$ be two dependencies in $\Sigma$. It is said that $\xi_1 \prec_P \xi_2$ if there exist instances $I$,$J$ and vector $\bar{a} \subseteq dom(J)$, such that:

1. $I \models \xi_2(\bar{a})$, and
2. there exists a trigger $(\xi_1, h)$, such that $I \xrightarrow{*,(\xi_1,h)} J$, and
3. $J \not\models \xi_2(\bar{a})$, and
4. there exists $X \in \bar{a} \cap \mathsf{Null}$ in the head of $\xi_2(\bar{a})$, such that null-pos$(\{X\}, I) \subseteq P$.

▶ **Example 54.** Consider $\Sigma$ containing a single tgd $\xi = R(x, y) \to \exists z\ R(y, z)$. In Section 3 we saw that there are instances $I$ such that standard-chase algorithm does not terminate on all branches for $I$ and $\Sigma$. It is easy to see that with instances $I = \{R(a, b)\}$ and $J = \{R(a, b), R(b, X)\}$, and vector $\bar{a} = (b, X)$, conditions 1,2 and 3 from the previous definition are fulfilled. For the 4th condition, consider $X \in \bar{a}$, then we have $\xi(\bar{a})$ which represents the formula $R(a, X) \to \exists z R(X, z)$. Thus, $X$ occurs in the head of $\xi(\bar{a})$. On the other hand, null-pos$(\{X\}, I) = \emptyset$, instance $I$ does not contain any labeled nulls, hence for any set $P$, null-pos$(\{X\}, I) \subseteq P$. Thus, $\xi \prec_P \xi$, for any set of positions $P$.

▶ **Definition 55.** [41] Let $P$ be a set of positions and $\xi$ a tgd. By aff-cl$(\xi, P)$ is denoted the set of positions $(R, i)$ from the head of $\xi$ such that one of the following holds:

1. for all $x \in Var^{\forall}(\xi)$[2], with $x$ occurs in $(R, i)$, $x$ occurs in the body of $\xi$ only in positions from $P$, or
2. position $(R, i)$ contains a variable $x \in Var^{\exists}(\xi)$.

For the tgd in Example 54, we have aff-cl$(\xi, P) = \{(R, 1), (R, 2)\}$, where $P = \{(R, 2)\}$. Given a set of dependencies $\Sigma$, the set of all positions in $\Sigma$ is written as $pos(\Sigma)$.

[2] Recall that by $Var^{\forall}(\xi)$ we denote the set of all universally quantified variables in $\xi$ and by $Var^{\exists}(\xi)$ the set of all existentially quantified variables in $\xi$.

▶ **Definition 56.** [41] A *2-restriction system* is a pair $(G(\Sigma), P)$, where $G(\Sigma)$ is a directed graph $(\Sigma, E)$ and $P \subseteq pos(\Sigma)$ such that:

1. for all $(\xi_1, \xi_2) \in E$, aff-cl$(\xi_1, P) \cap pos(\Sigma) \subseteq P$ and aff-cl$(\xi_2, P) \cap pos(\Sigma) \subseteq P$, and
2. for all $\xi_1 \prec_P \xi_2$, $(\xi_1, \xi_2) \in P$.

A 2-restriction system is *minimal* if it is obtained from $((\Sigma, \emptyset), \emptyset)$ by a repeated application of constraints 1 and 2, from the previous definition, such that $P$ is extended only by those positions that are required to satisfy condition 1. Let us denote by $part(\Sigma, 2)$ the set that contains the sets of all strongly connected components in a minimal 2-restriction system.

▶ **Example 57.** Returning to our dependency from Example 54, the minimal 2-restriction system is computed as follows. Consider pair $((\{\xi\}, \emptyset), \emptyset)$. Previously we showed that $\xi \prec_P \xi$, for any set of positions $P$, by particularization we have $\xi \prec_\emptyset \xi$. Thus, we add edge $(\xi, \xi)$ to $E$. Using condition 1 from Definition 56 we have aff-cl$(\xi, \emptyset) = \{(R, 2)\}$. That is we add position $(R, 2)$ to $P$. By repeating this process once again with $P = \{(R, 2)\}$, we add to $P$ the position $(R, 1)$ too. Hence, the minimal 2-restriction system is $((\Sigma, \{(\xi, \xi)\}, \{(R, 1), (R, 2)\}\})$. The only connected component in this restriction system is $\{\xi\}$.

In [41], Meier et al. provide a simple algorithm that computes the set $part(\Sigma, 2)$.

▶ **Definition 58.** [41] A set $\Sigma$ of tgds is called *inductively restricted* iff every $\Sigma' \in part(\Sigma, 2)$ is safe. The set of all inductively restricted tgd sets is denoted by $\mathsf{IR}$.

Using the same reduction from the proof of Theorem 48, it can be shown that the problem of testing if $\xi_1 \prec_P \xi_2$, for a given $\xi_1$, $\xi_2$ and $P$, is coNP-hard and it can be solved in $\Sigma_2^P$. Similarly to the stratification case it can be shown that the complexity of testing if $\Sigma \in \mathsf{IR}$ is in $\Pi_3^P$. From the definition, it directly follows that $\mathsf{SD} \subset \mathsf{IR}$. To this Meier et al. [41] also showed that $\mathsf{Str} \nparallel \mathsf{IR}$ and that $\mathsf{CStr} \subset \mathsf{IR}$. On the other hand, the classes $\mathsf{SwA}$ and $\mathsf{IR}$ are incomparable [47], that is $\mathsf{SwA} \nparallel \mathsf{IR}$.

▶ **Example 59.** [41] Consider the following set of tgds $\Sigma$:

$$\begin{aligned} \xi_1 &= S(x), E(x, y) \to E(y, x), \text{ and} \\ \xi_2 &= S(x), E(x, y) \to \exists z\, E(y, z), E(z, x). \end{aligned}$$

It can be easily observed that $\Sigma$ is neither stratified nor safe, but it is inductively restricted.

▶ **Theorem 60.** [41] *Let $\Sigma \in \mathsf{IR}$ and let $I$ be an instance. Then there exists a polynomial, in size of $I$, that bounds the length of every standard-chase sequence of $I$ and $\Sigma$.*

Meier, Schmidt and Lausen [41] observed that the inductive restriction criterion can be extended to form a hierarchy of classes that ensure the standard-chase termination on all branches for all instances. Intuitively, the lowest level of this hierarchy, noted $T[2]$, is the class of inductively restricted dependencies. Level $T[k]$, $k > 2$ is obtained by extending the binary relation $\prec_P$ to a $k$-ary relation $\prec_{k,P}$. Intuitively, $\prec_{k,P}$ $(\xi_1, \ldots, \xi_k)$ means that there exists a standard-chase sequence such that firing $\xi_1$ will also cause $\xi_2$ to fire. This in turn will cause $\xi_3$ to fire and so on until $\xi_k$. Based on this new relation, the set $part(\Sigma, k)$ is computed similarly to $part(\Sigma, 2)$. The algorithm that computes $part(\Sigma, k)$ can be found in [41]. For all $k \leq 2$, it is shown that $T[k] \subset T[k+1]$.

It is easy to check that the previous hierarchy is closed under semi-enrichment, following from Corollary 27 that for any $k$ we have $T[k] \subset \mathsf{CT}_{\forall\forall}^{\mathsf{sobl}}$. Also in [44] it is shown that $T[k] \nparallel \mathsf{CT}_{\forall\forall}^{\mathsf{obl}}$. More recently, the $T[k]$ hierarchy of classes was extended by Meier et al. [42]

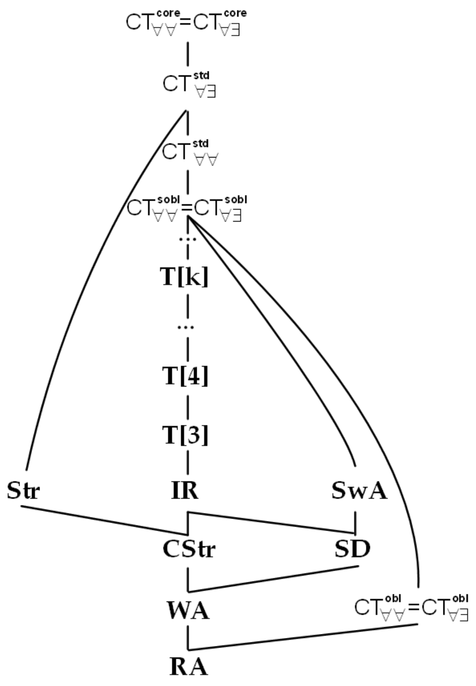

**Figure 5** Relationship between chase termination classes.

to the $\forall\exists - T[k]$ hierarchy of classes that ensures the standard-chase termination on at least one execution branch and it showed that $T[k] \subset \forall\exists - T[k]$, for any $k > 1$.

Figure 5 illustrates, as a Hasse diagram, the subset relationship between the termination classes presented.

Before concluding this subsection, we need to mention that more recently Greco et al. [27] extended the classes of dependencies that ensure the standard-chase termination to new large classes based on a stratification based method called local stratification.

## 4.7 The rewriting approach

Spezzano and Greco [47] noticed that all the previous classes may be extended by using a rewriting technique. Intuitively, if $\mathbf{T}$ is one of the classes $\{\mathsf{WA}, \mathsf{SD}, \mathsf{SwA}, \mathsf{Str}, \mathsf{CStr}\}$, then instead of directly checking if a set of dependencies $\Sigma \in \mathbf{T}$, we check if $\text{Adn}(\Sigma) \in \mathbf{T}$, where $\text{Adn}(\Sigma)$ is an adornment based rewriting of $\Sigma$ such that, if $\text{Adn}(\Sigma) \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, then $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. Where the adornment of a predicate $p$ of arity $m$ is a string of the length $m$ over the alphabet $\{b, f\}$. An adorned atom is of the form $p^{\alpha_1,\alpha_2,\ldots,\alpha_m}(x_1, x_2, \ldots, x_m)$; if $\alpha_i = b$, then variable $x_i$ is considered bounded, otherwise the variable is considered free.

Due to the space constraints we will present this method following a simple example.

Consider the following set of dependencies $\Sigma = \{\xi_1, \xi_2\}$ [47]:

$$\begin{array}{rcl}\xi_1 & = & N(x) \to \exists y\, E(x,y), \text{ and} \\ \xi_2 & = & S(x), E(x,y) \to N(y).\end{array}$$

The affected positions in $\Sigma$ are $(E,1)$,$(E,2)$ and $(N,1)$. As the corresponding propagation graph contains a cycle, through an existential edge, involving positions $(N,1)$ and $(E,2)$, it follows that $\Sigma \notin \mathsf{SD}$. Construct the set of dependencies $\text{Adn}(\Sigma)$ as follows:

1. For all predicate symbols $p$ of arity $m$ in $\Sigma$ add the tgd:

$$\forall x_1, x_2, \ldots, x_m \; p(x_1, x_2, \ldots, x_m) \rightarrow p^{\alpha_1, \alpha_2, \ldots, \alpha_m}(x_1, x_2, \ldots, x_m)$$

where, for all positive $i \leq m$, $\alpha_i = b$.
In our example $\Sigma$ contains the following predicate symbols $\{E, S, N\}$, that is we add to Adn($\Sigma$) the following set of tgds:
$\xi'_1 \; = \; E(x, y) \rightarrow E^{b\,b}(x, y);$
$\xi'_2 \; = \; N(x) \rightarrow N^{b}(x);$ and
$\xi'_3 \; = \; S(x) \rightarrow S^{b}(x).$

2. Repeat to create new adornment predicate symbols based on the existing dependencies, until none can be added. That is, if a variable in the head is marked as bounded (free) and if it occurs only bounded (free) places in the body. All existential variables in the head are marked as free.
Returning to our example and using $\xi_1$ from $\Sigma$ and the new adornment $N^b$, we add the following dependency to Adn($\Sigma$):
$\xi'_4 \; = \; N^{b}(x) \rightarrow \exists y \; E^{b\,f}(x, y).$
Similarly, based on tgd $\xi_2$ from $\Sigma$ and new adornments $S^b$ and $E^{b\,b}$, we add the following dependency to Adn($\Sigma$):
$\xi'_5 \; = \; S^{b}(x), E^{b\,b}(x, y) \rightarrow N^{b}(y).$
Repeating this process, we add the following tgds to Adn($\Sigma$):
$\xi'_6 \; = \; S^{b}(x), E^{b\,f}(x, y) \rightarrow N^{f}(y),$ and
$\xi'_7 \; = \; N^{f}(x) \rightarrow \exists y \; E^{f\,f}(x, y).$
After this point no other adornments can be created.

3. Finally, for each of the adornment predicate $p^\alpha$ in Adn($\Sigma$), add a new dependency in Adn($\Sigma$) that "copies" $p^\alpha$ to a new $\hat{p}$ predicate symbol. In this example the following new dependencies are added:
$\xi'_8 \; = \; N^{b}(x) \rightarrow \hat{N}(x);$
$\xi'_9 \; = \; N^{f}(x) \rightarrow \hat{N}(x);$
$\xi'_{10} \; = \; S^{b}(x) \rightarrow \hat{S}(x);$
$\xi'_{11} \; = \; E^{b\,b}(x, y) \rightarrow \hat{E}(x, y);$
$\xi'_{12} \; = \; E^{b\,f}(x, y) \rightarrow \hat{E}(x, y);$ and
$\xi'_{13} \; = \; E^{f\,f}(x, y) \rightarrow \hat{E}(x, y).$

In [47], it is proved that $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ if and only if Adn($\Sigma$) $\in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. Returning to our example, it can be noted that the set Adn($\Sigma$) is safe. Thus, using the previous observation, it results that even if the set $\Sigma$ was not safe, the standard chase will terminate on all branches on $\Sigma$ with any instances.

▶ **Theorem 61.** [41] *Let* $\mathbf{T}$ *be one of the classes* $\{\mathsf{WA}, \mathsf{SD}, \mathsf{SwA}, \mathsf{Str}, \mathsf{CStr}\}$*, let* $\Sigma$ *be a set of tgds and let* $I$ *be an instance. Then, if Adn*($\Sigma$) $\in \mathbf{T}$*, there exists a polynomial, in size of* $I$*, that bounds the length of every standard-chase sequences of* $I$ *and* $\Sigma$*.*

Even more, Spezzano and Greco [47] proved that these rewritings strictly extend the classes of dependencies.

▶ **Theorem 62.** [41] *Let* $\mathbf{T}$ *be one of the classes* $\{\mathsf{WA}, \mathsf{SD}, \mathsf{SwA}, \mathsf{Str}, \mathsf{CStr}\}$ *and let denote by* $\mathsf{Adn}\mathbf{T}$ *the set of all* $\Sigma$ *such that Adn*($\Sigma$) $\in \mathbf{T}$*. Then,* $\mathbf{T} \subset \mathsf{Adn}\mathbf{T}$*.*

More recently, this rewriting method was further improved by Greco et al. [27] by indexing the adornment used to specify the free positions. This method ensures that we may equate only variables that have the adornment with the same index.

## 5 The chase and data exchange

The previous section was mainly focused on presenting different chase algorithms and their termination criteria. This section is dedicated to the instance returned by the chase algorithm and to how it can be used in the data-exchange problem. As we will see, the finite instance returned by any chase variation is strongly related to the notion of *universal model*. Such instances represent a good candidate to be materialized under the target schema in data exchange. Beside computing a general solution for the data-exchange problem, the chase procedure also plays an important role in some related problems as the inverse, recovery [18, 22, 8], and composition of schema mappings [21, 6].

For a complete and coherent introduction to the application of the chase procedure in data exchange, we first present the notion of universal models and its relation with the chase algorithm. Need to mention that the notion *universal models* [16] was introduced as a generalization of *universal solutions* [19] in data exchange. This first part will be followed by a short review of the data-exchange problem and the link between universal models and query answering in data exchange. In the final part of this section we will review the query answering problem in case there are no universal models.

### 5.1 Universal models

Beside the data-exchange problem, universal models play an important role in many other database problems as: testing for conjunctive query containment under functional and inclusion dependencies [31], data integration [33], and query answering over ontologies [14].

▶ **Definition 63.** [16] Given an instance $I$ and $\Sigma$ a set of dependencies, a finite instance $J$ is said to be a *model* for $I$ and $\Sigma$ if $J \models \Sigma$, and $I \to J$.

▶ **Example 64.** Consider $I = \{R(a,b), R(b,c)\}$ and $\Sigma = \{R(x,y), R(y,z) \to R(x,z)\}$. The instance $J = \{R(a,b), R(b,c), R(a,c)\}$ is a model of $I$ and $\Sigma$, so is instance $J_1 = J \cup \{R(a,X)\}$, with $X$ a labeled null from Null. On the other hand, $J_2 = \{R(a,b), R(a,c)\}$ is not a model of $I$ and $\Sigma$, even if $J_2 \models \Sigma$, since there is no homomorphism from $I$ into $J_2$.

The conclusion of this example is that, in general, there may be an infinite number of models of $I$ and $\Sigma$. Still, some of these models are more general than the others in the sense that they have a homomorphism into all the other models. Such models are called, of course, universal models.

▶ **Definition 65.** [16] A finite instance $U$ is said to be a *weak universal model* of $I$ and $\Sigma$ if $U$ is a model of $I$ and $\Sigma$, and if for any finite model $J$ of $I$ and $\Sigma$, it is that $U \to J$. If $U \to J$. Also for all infinite models $J$ of $I$ and $\Sigma$, then $U$ is said to be a *strong universal model* or simply a *universal model*.

▶ **Example 66.** Considering the instance $I$ and the dependency $\Sigma$ from Example 64, it is clear that both instances $J$ and $J_1$ are strong universal models. Moreover, the model $J_3 = \{R(a,b), R(b,c), R(a,c), R(a,a)\}$ is neither a strong nor weak universal model as there does not exist a homomorphism from $J_3$ to model $J$.

▶ **Theorem 67.** [19, 16] *Let $I$ be an instance and $\Sigma$ a set of tgds and egds. Then any finite instance returned by the standard-chase algorithm is a universal model of $I$ and $\Sigma$.*

Intuitively, the theorem says that whenever the standard chase terminates and it does not fail, it gives a universal model of $I$ and $\Sigma$. From this theorem, it follows that if $chase^{\mathbf{std}}{}_{\Sigma}(I) \neq \bot$ then $chase^{\mathbf{std}}{}_{\Sigma}(I)$ is a universal model for $I$ and $\Sigma$. In the finite case, the instance returned by the standard-chase algorithm is homomorphically equivalent with the finite result of any chase variations. It follows that for any of the previously presented chase variations, when they terminate and do not fail, they return a universal model.

▶ **Corollary 68.** *Let $I$ be an instance, $\Sigma$ a set of tgds and egds, and $* \in \{\mathbf{obl}, \mathbf{sobl}, \mathbf{core}\}$. If $chase^{*}{}_{\Sigma}(I) \neq \bot$, then $chase^{*}{}_{\Sigma}(I)$ is a universal model of $I$ and $\Sigma$.*

This result ensures that the standard, oblivious, semi-oblivious and core chase are sound in finding universal models. Naturally the following question raises: *Are these algorithms also complete in finding universal models?* The following example shows that the standard, oblivious and semi-oblivious-chase algorithms are not complete.

▶ **Example 69.** Let us consider the same instance $I = \{R(a, b)\}$ and set $\Sigma = \{\xi_1, \xi_2\}$, where:

$$\begin{aligned}\xi_1 &= R(x, y) \to \exists z\, R(y, z), \text{ and}\\ \xi_2 &= R(x, y), R(y, z) \to R(y, y).\end{aligned}$$

It is easy to see that there is no terminating branch for the standard chase for $\Sigma$ and $I$. Similarly, the oblivious and semi-oblivious algorithms with the same input will not terminate. On the other hand, there exists universal model $J = \{R(a, b), R(b, b)\}$ of $I$ and $\Sigma$. Thus the standard chase is not complete in finding universal models.

The result below shows that the core chase is complete in finding universal models.

▶ **Theorem 70.** [16] *Let $I$ be an instance and $\Sigma$ a set of tgds and egds. Then there exists a universal model of $I$ and $\Sigma$ iff the core-chase algorithm terminates and does not fail on input $I$ and $\Sigma$.*

We know from the definition of the universal models that all universal models are also weak universal models. The following example shows that the converse does not hold.

▶ **Example 71.** [16] Let us consider instance $I = \{T(a)\}$ and $\Sigma = \{\xi_1, \xi_2, \xi_3\}$, where:

$$\begin{aligned}\xi_1 &= T(x) \to \exists y, z\, E(y, z);\\ \xi_2 &= E(x, y) \to \exists z\, E(y, z); \text{ and}\\ \xi_3 &= E(x, y), E(y, z) \to E(x, z).\end{aligned}$$

Consider the relation $E$ to contain the edges of a graph. Clearly all models have an infinite walk. From this it follows that every finite model has a cycle in the corresponding graph. From this and $\xi_3$, it also follows that any finite model has a self loop. Besides, the instance $J = \{T(a), E(X, X)\}$ is a model of $I$ and $\Sigma$ containing a self loop. Consequently, $J$ is a weak universal model of $I$ and $\Sigma$. On the other hand, the transitive closure of an infinite path also satisfies $\Sigma$, however no finite instance with cycle has a homomorphism into it. This means that $J$ is not a strong universal model of $I$ and $\Sigma$.

Deutsch et al. [16] showed that it is undecidable to test if an instance $U$ is a strong (weak) universal model for a given instance $I$ and $\Sigma$ a set of tgds. Even more, they demonstrated that there is no complete chase based procedures for finding weak universal models.

## 5.2 Data exchange

Data exchange is an old database problem that only recently earned more formal treatment. More precisely, it is the problem of transforming data structured under a source schema to data structured under a different target schema. Formally, a data-exchange setting is a quadruple $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S}$ represents the source schema, $\mathbf{T}$ represents the target schema, $\Sigma_{st}$ is a set of constraints representing the relationship between the source and target schema, and $\Sigma_t$ represents a set of constraints over the target schema. Given a data-exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ and the instance $I$ over the source schema $\mathbf{S}$, the data-exchange problem is to find instances $J$ over the target schema $\mathbf{T}$, such that $I \cup J$ is a model for $I$ and $\Sigma_{st} \cup \Sigma_t$. An instance $J$ with the previous properties is called a *solution* to the data-exchange problem, or simply a solution. This problem was first formalized by Fagin et al. in [19]. Most of the data-exchange problems consider $\Sigma_{st}$ to be a set of tgds and $\Sigma_t$ to be a set of tgds and egds. From now on, if not mentioned otherwise, we assume that the data-exchange settings are of this format.

As there is an infinite number of solutions to the data-exchange problem, a natural question raises: *Which solution or finite set of solutions should be materialized on the target?* There is no simple answer to this question as there may be different representations of the target depending on the semantics of the queries used over the target instance. The semantics considered in this subsection, also most prominent in the literature, is the certain answer semantics for union of conjunctive queries (UCQ) over the target instance. This can be formalized by the following definition:

▶ **Definition 72.** Let $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, let $I$ be a source instance and $Q$ a query in UCQ over $\mathbf{T}$. The certain answer of $Q$ for $I$ and $\sigma$ is defined as

$$cert_\sigma(Q, I) =^{def} \bigcap_{J,\ I\cup J \models \Sigma_{st}\cup\Sigma_t} Q(J).$$

Fagin et al. [19] showed that the *universal solution* is a good candidate to be materialized in data-exchange problem under the certain UCQ answer semantics. Where the universal solution for a data-exchange setting $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ and the instance $I$ is a universal model for $I$ and $\Sigma_{st} \cup \Sigma_t$ restricted to schema $\mathbf{T}$. We need to mention that Fagin et al. considered as solutions only finite instances which is the more important case in practice. This means that all results specified in Subsection 5.1 also hold for universal solutions. In particular, it means that the universal solution can be computed by the chase algorithms and that it is undecidable if the universal solution exists for a given data-exchange setting and a given source instance. Marnette [38] showed that it is undecidable to test if the oblivious chase will terminate for a given data-exchange setting for all input instances. This result can be enhanced to all chase variations, including core chase. Thus, it is undecidable to test if, for a given data-exchange setting for all the input instances, there exists a universal solution.

In [19], Fagin et al. described a sufficient condition for the universal solution to not exist, as the following theorem shows it:

▶ **Theorem 73.** [19] *Let $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data-exchange setting and $I$ a source instance such that there is a failing branch for the standard chase with input $I$ and $\Sigma_{st} \cup \Sigma_t$. Then there is no universal solution for $I$ and $\sigma$.*

In data exchange we may also have the case when there exists a solution but there is no universal solution. Let us consider the next example:

▶ **Example 74.** Consider the following data-exchange setting:

$$\sigma = (\{S\}, \{R\}, \{S(x,y) \to R(x,y)\}, \{R(x,y) \to \exists z\ R(y,z)\})$$

and the source instance $I = \{S(a,b)\}$. Clearly there is no universal solution for this setting, but there exists solution $J = \{R(a,b), R(b,b)\}$.

As shown by Kolaitis et al. in [32], it is undecidable to check for a given instance $I$ and a data-exchange setting $\sigma$, if there exists a solution for $I$ and $\sigma$.

Before presenting the computation of the certain answer for a data-exchange setting using a universal model, we need to introduce the notion of naïve evaluation. Let $I$ be an instance, possible with null values, and $Q$ be a query. The $Q_{\text{naïve}}(I)$ is defined by evaluating $Q$ on $I$ and by treating each null as a new distinct constants, and then by eliminating from the result all the tuples with nulls.

▶ **Theorem 75.** [19] *Let $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data-exchange setting and $I$ an instance over the source instance that does not contain nulls such that there exists a universal solution $J$ for $I$ and $\sigma$. Then, $cert_\sigma(Q, I) = Q_{naïve}(J)$ for any $Q \in$* UCQ.

In [35] Libkin showed that UCQ is the largest class of queries with the property that the certain answers may be computed using the naïve evaluation. We conclude this subsection by reiterating the idea that within the infinite set of universal solutions there exists a universal solution which is minimal in size. Such universal solution is called core and, as noted in [20], it is unique up to variable renaming. Hence, in case there exists a universal solution, the core chase will terminate and return the core.

## 5.3 Data exchange beyond universal solutions

For the data-exchange setting and the source instance presented in Example 74 we know that there is no universal solution. On the other hand, when considering the query $Q(x) \leftarrow \exists y\ R(x,y)$, the certain answers is the set of tuples $\{(a), (b)\}$. In [12], Cali, Gottlob and Kifer investigate the problem of conjunctive query answering when the universal solution is not guaranteed to exist. For this, the authors unravel two classes of tgds , namely *guarded tuple generating dependencies* (gtgd) and *weakly guarded tuple generating dependencies* (wgtgd), for which the problem of conjunctive query evaluation is decidable. Intuitively, a tgd is guarded if its body contains an atom called *guard* which covers all the variables in the body. Clearly LAV tgds are gtgds. A set of tgds is weakly guarded, if for each tgd, its body contains one atom which covers all the variables that appear in the affected position, that is, predicate positions that may contain new labeled nulls generated during the chase process.

▶ **Example 76.** Let us consider the following dependencies:

$$\begin{aligned} \xi_1 &= S(x), R(x,y) \to \exists z\ R(y,z), \text{ and} \\ \xi_2 &= R(x,z), R(z,y) \to R(y,x). \end{aligned}$$

In $\xi_1$, the atom $R(x,y)$ covers all the variables in the body, meaning that it is a gtgd. Clearly, $\xi_2$ is not gtgd as there is no atom to cover all variables from the body. The affected position in the set $\{\xi_1, \xi_2\}$ is $(R, 2)$, that is we may introduce new labeled nulls during the chase process only in the second position of the predicate $R$. As in $\xi_2$, the atom $R(z,y)$ covers both variables that appear in affected position in $\xi_2$. It follows that $\xi_2$ is a wgtgd.

Cali et al. [12] give complexity bounds for the conjunctive query answering problem, that is: *Does a tuple $t$ belong to the certain answer?* The complexity bounds discovered are the following: (1) for a fixed set gtgds the conjunctive query answering problem is NP-complete; (2) for atomic queries the problem becomes polynomial; (3) in case the fixed dependencies are wgtgds, the conjunctive query answering problem becomes EXPTIME-complete. Need to mention here that in [29] Hernich showed that if the data-exchange setting contains only guarded tgds, it is decidable if for the given setting and a given instance there exists a universal solution.

In the certain answer semantics for UCQ queries over the target schema a universal solution is enough to compute certain answer for any UCQ query. Therefore another question comes up naturally: *Is this semantics also a good model for general queries?* As shown in [4] and [34], this semantics is not suitable for general queries, as it may give unintuitive answers even for simple copying data-exchange settings.

▶ **Example 77.** Let us consider a data-exchange setting $\sigma = (\{R\}, \{R'\}, \Sigma_{st}, \emptyset)$, where $\Sigma_{st}$ simply copies the source into target: $R(x, y) \to R'(x, y)$. Consider the source instance $I = \{R(a, b)\}$ and the query over the target schema $Q(x, y) \leftarrow R'(x, y) \wedge \neg R'(x, x)$. As one of the solution is the instance $J = \{R'(a, b), R'(a, a)\}$, it follows that $cert_\sigma(Q, I) = \emptyset$. Now, when applying the same query on the source instance (by replacing relation name $R'$ with $R$), it returns the set of tuples $\{(a, b)\}$. Clearly this is not the expected behavior as the target instance is supposed to be a copy of the source instance.

To avoid such cases, Fagin et al. [20] proposed a new semantics for the certain answers to existential queries. Where existential queries $Q(\bar{x})$ is a formula of the form $\exists \bar{y}\ \varphi(\bar{x}, \bar{y})$, where $\varphi$ is a *safe* quantifier-free formula. Under this semantics, instead of evaluating the query on all solutions, the query is evaluated on universal solutions only.

▶ **Definition 78.** Let $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, let $I$ be a source instance and $Q$ a query over the schema $\mathbf{T}$. The *u-certain* answer of $Q$ for $I$ and $\sigma$ is defined as

$$u\text{-}cert_\sigma(Q, I) =^{def} \bigcap \{Q(J) \ : \ J \text{ universal solution for } I \text{ and } \sigma\}.$$

Clearly $cert_\sigma(Q, I) \subseteq u\text{-}cert_\sigma(Q, I)$, for any data-exchange setting $\sigma$, instance $I$ and query $Q$. Also, as shown in [20], $cert_\sigma(Q, I) = u\text{-}cert_\sigma(Q, I)$ whenever $Q \in \mathsf{UCQ}$. The u-certain semantics is shown [20] to be adequate for existential queries. Even more, it is proved that in case $J$ is a universal solution for data-exchange setting $\sigma$ and instance $I$, and $Q$ is an existential query, then the answer under u-certain semantics can be computed as: $u\text{-}cert_\sigma(Q, I) = Q_{\text{naïve}}(core(J))$. Returning to the previous example, the core universal solution is $J = \{R(a, b)\}$, hence the certain answer to query $Q$ will be the expected set of tuples $\{(a, b)\}$.

Later on new closed world semantics was proposed in order to deal with general queries for the data-exchange problem [34, 30, 28, 24]. Libkin [34] considered data-exchange settings without target dependencies and computed CWA-solutions which are used afterwards to compute certain answers for FO queries. Hernich and Schweikardt [30] introduced a new chase based algorithm, called the $\alpha$-chase, to compute CWA-solutions when the data-exchange setting also contains target dependencies. In [24] Grahne and O. introduce a chase algorithm on conditional tables in order to strongly represent a closed world semantics called the constructible solutions. A similar chase process for conditional tables that considers only source to target dependencies was also introduced in [7].

## 6 Chase extensions

The chase algorithms presented in the previous sections considered only tgds and egds as constraints. In this section we will describe extensions of the chase algorithms needed in order to deal with *negation disjunctive embedded dependencies* (NDED). As we will see, the chase procedure on NDEDs helps finding universal solution sets which are used afterwards in computing certain answers to more general queries such as $\mathsf{UCQ}^{\neg,\neq}$. Disjunctive dependencies are also investigated by Marnette and Geerts in [40].

Before introducing the chase process for NDEDs, we need to extend the universal solution notion to *universal solution set*. Given two instances $I$ and $J$, we write $I \dashrightarrow J$ if there exists an embedding from $I$ to $J$. Let $\mathcal{I}$, $\mathcal{J}$ be two sets of instances, we write $\mathcal{I} \dashrightarrow \mathcal{J}$ if for all $J \in \mathcal{J}$ there exists $I \in \mathcal{I}$ such that $I \dashrightarrow J$.

▶ **Definition 79.** [16] A set $\mathcal{I}$ of finite instances is an *emb-universal model set* for a set of instances $\mathcal{J}$ if it satisfies the following conditions:

1. $\mathcal{I} \dashrightarrow \mathcal{J}$.
2. $\mathcal{I} \subseteq \mathcal{J}$.
3. $\mathcal{I}$ is finite.
4. there is no $\mathcal{I}' \subset \mathcal{I}$ such that $\mathcal{I}' \dashrightarrow \mathcal{J}$.

Let us first review the extended chase step for disjunctive embedded dependencies. A disjunctive embedded dependency (DED) [15] is a constraint of the form:

$$\xi: \ \forall \bar{x}\ \alpha(\bar{x}) \rightarrow \bigvee_{1\leq i\leq n} \exists \bar{z}_i\ \beta_i(\bar{x}_i, \bar{z}_i)$$

where, $\bar{x}_i \subseteq \bar{x}$, for every $1 \leq i \leq n$. Formulae $\alpha$ and each $\beta_i$ are conjunctions of relational symbols and equality atoms. For each $1 \leq i \leq n$, let us denote by $\xi_i$ the dependency $\forall \bar{x}\ \alpha(\bar{x}) \rightarrow \exists \bar{z}_i\ \beta_i(\bar{x}_i, \bar{z}_i)$. The extended chase step on DED is defined as follows [16]. Let $I$ be an instance and a homomorphism $h$ such that $h(\alpha(\bar{x})) \subseteq I$. If $I \xrightarrow{(\xi_i,h)} \perp$, for all $1 \leq i \leq n$, then we say that the extended chase step on $I$ with $(\xi, h)$ *failed*, and it is denoted as $I \xrightarrow{(\xi,h)} \perp$. Otherwise, let $\mathcal{J}$ be the set containing all instances $J_i$, such that $I \xrightarrow{(\xi_i,h)} J_i$. If $\mathcal{J}$ is empty, it is said that the extended chase step on $I$ with $(\xi, h)$ is *not applicable*. For convenience we write this as $I \xrightarrow{(\xi_i,h)} I$. Finally, if $\mathcal{J} \neq \emptyset$, then $\mathcal{J}$ is said to be obtained from $I$ in one extended chase step with $(\xi, h)$ and denoted as $I \xrightarrow{(\xi,h)} \mathcal{J}$

▶ **Example 80.** Consider the following DED:

$$\xi \ = \ R(x, y), R(y, z) \rightarrow R(x, z) \vee x = y \vee \exists v\ R(v, z).$$

Let $I = \{R(a, b), R(b, c)\}$. Let $h = \{x/a, y/b, z/c\}$ be the homomorphism that maps the body of $\xi$ to $I$. The three disjuncts from the head of $\xi$ give the following dependencies:

$$\begin{aligned}
\xi_1 &= R(x, y), R(y, z) \rightarrow R(x, z);\\
\xi_2 &= R(x, y), R(y, z) \rightarrow x = y;\ \text{and}\\
\xi_3 &= R(x, y), R(y, z) \rightarrow \exists v\ R(v, z).
\end{aligned}$$

For these dependencies, we have $I \xrightarrow{(\xi_1,h)} J$, where $J = I \cup \{R(a, c)\}$, $I \xrightarrow{(\xi_2,h)} \perp$ and $I \models \xi_3$. Thus $I \xrightarrow{(\xi,h)} \mathcal{J}$, where $\mathcal{J} = \{J\}$.

A dependency $\xi$ of the form $\alpha(\bar{x}) \to \bot$, where $\alpha$ is a conjunction of atoms, is called *denial constraint* or *falsehood* . If for an instance $I$ there exists a homomorphism $h$, such that $h(\alpha(\bar{x})) \subseteq I$, then it is said that the extended chase *failed* on $I$ with $(\xi, h)$ and it is denoted by $I \xrightarrow{(\xi,h)} \bot$.

If we add inequalities to DED$s$, we obtain DED$^{\neq}s$ [15]. Let $\Sigma$ be a set of DED$^{\neq}s$ over the schema $\mathbf{R}$. The set of dependencies $\Sigma$ is replaced by $\Sigma^{\neq}$, in which each inequality of the from $x \neq y$ from $\Sigma$ is replaced by the atom $N(x, y)$, where $N$ is a new predicate which does not appear in $\Sigma$. Also in $\Sigma^{\neq}$ are added the following dependencies: $\to x = y \vee N(x, y)$, and $x = y \wedge N(x, y) \to \bot$. It may be noticed that in the new schema, $\Sigma^{\neq}$ contains one extra predicate compared to the schema of $\Sigma$ and also it contains two new dependencies.

Finally, by adding to DED$^{\neq}s$ negation we obtain NDED$s$. Let $\Sigma$ be a set of NDED$s$ over the schema $\mathbf{R}$. By $\Sigma^{\neq,\neg}$ is denoted the set of dependencies $\Sigma^{\neq}$ in which each negated literal of the form $\neg R(\bar{x})$ is replaced by a new literal $\hat{R}(\bar{x})$, and also for each predicate $R \in \mathbf{R}$ the following two dependencies are added in $\Sigma^{\neq,\neg}$: $R(\bar{x}) \vee \hat{R}(\bar{x})$, and $R(\bar{x}) \wedge \hat{R}(\bar{x}) \to \bot$. It can be noted that for any set $\Sigma$ of NDED, $\Sigma^{\neq,\neg}$ is a set of DED.

▶ **Example 81.** Consider the following set of dependencies $\Sigma = \{\xi_1, \xi_2\}$, where:

$$\begin{aligned} \xi_1 &= R(x, y) \to x \neq y, \text{ and} \\ \xi_2 &= R(x, y), S(x) \to \neg S(y). \end{aligned}$$

The corresponding $\Sigma^{\neq,\neg}$ will contain the dependencies:

$$\begin{aligned} \xi_1 &= R(x, y) \to N(x, y); \\ \xi_2 &= R(x, y), S(x) \to \hat{S}(y); \\ \xi_3 &= x = y \vee N(x, y); \\ \xi_4 &= x = y \wedge N(x, y) \to \bot; \\ \xi_3 &= R(x, y) \vee \hat{R}(x, y); \\ \xi_4 &= R(x, y) \wedge \hat{R}(x, y) \to \bot; \\ \xi_5 &= S(x) \vee \hat{S}(x); \text{ and} \\ \xi_6 &= S(x) \wedge \hat{S}(x) \to \bot. \end{aligned}$$

Using the previous notations we are now ready to present the *extended-core-chase* algorithm introduced by Deutsch, Nash and Remmel in [16] which has as input an instance $I$ and a set $\Sigma$ of NDED.

```
EXTENDED-CORE-CHASE(I,Σ)
1 $\mathcal{L}_0 = \{I\}$; $i := 0$;
2 Compute in parallel for each instance $J \in \mathcal{L}_i$ the set $\mathcal{K}_J$
  where $K \in \mathcal{K}_J$ iff $J \xrightarrow{(\xi,h)} K$ for some $\xi \in \Sigma^{\neq,\neg}$ and homomorphism $h$
3 $\mathcal{L}' = \bigcup_{J \in \mathcal{L}_i} \bigcup_{K \in \mathcal{K}_J} \{core(K)\}$
4 compute $\mathcal{L}_{i+1}$ by removing from $\mathcal{L}'$ all $K$ such that $\exists L \in \mathcal{L}'$, $L \to K$; i = i +1;
5 if $\mathcal{L}_i = \mathcal{L}_{i-1}$
6    then return the set of instances from $\mathcal{L}_i$ restricted to the schema of $I$
7    else goto 2
```

▶ **Example 82.** Consider $\Sigma = \{T(x) \to R(x)\}$ over the schema $\{R, S, T\}$ and consider the instance $I = \{T(a)\}$ over the same schema. With this input, the value of $\mathcal{L}_1$ after executing step 4 is $\mathcal{L}_1 = \{\{T(a), R(a), S(a)\}, \{T(a), R(a), \hat{S}(a)\}\}$, thus the algorithm will return the set $\{\{T(a), R(a), S(a)\}, \{T(a), R(a)\}\}$.

Let us denote by $\Sigma(I)$ the set of all models for $I$ and $\Sigma$. The following theorem, due to [16], ensures that the returned set of instances, if it terminates, is an emb-universal model set for the set of all models of $I$ and $\Sigma$.

▶ **Theorem 83.** [16] *Let $\Sigma$ be a set of NDEDs over the schema $\mathbf{R}$ and let $I$ be an instance over the same schema, such that the extended-core-chase algorithm terminates with the input $I$ and $\Sigma$ returning the set of instances $\mathcal{L}$. Then $\mathcal{L}$ is an emb-universal model set for $\Sigma(I)$.*

As shown in [16], emb-universal model sets can be used to compute the certain answers to $\mathsf{UCQ}^{\neq,\neg}$.

▶ **Theorem 84.** [16] *Let $\mathcal{U}$ be a emb-universal model set for $\Sigma(I)$, and let $Q$ be a $\mathsf{UCQ}^{\neq,\neg}$ query. Then $cert_\Sigma(Q, I) = \bigcap_{J\in\mathcal{U}} Q(J)$.*

Let us consider the dependencies and the instance from Example 82 and also consider the query $Q(x) \leftarrow R(x) \wedge \neg S(x)$. When computing query $Q$ against the emb-universal model set from Example 82, $cert_\Sigma(Q, I) = \emptyset$. The previous result does not hold for general FO queries. For this consider the boolean query $Q' \leftarrow (\forall x\ S(x) \rightarrow R(x))$. In this case $Q'(J)$ is **true** for all instance $J$ from the emb-universal model set. On the other hand, the instance $J = \{T(a), R(a), S(b)\}$ is a model for $I$ and $Q(J) =$ **false**, that is $cert_\Sigma(Q, I) =$ **false**. In order to cope with general FO queries in data exchange, several closed world semantics have been proposed [34, 30, 28, 24].

## 7 Conclusion

This chapter was intended to be a review of the chase based algorithms and also to highlight their use in data exchange. One of the main issues with the chase algorithms is the termination problem, that is:

- *Is there a branch for which the algorithm terminates for a given input $I$ and $\Sigma$?*
- *Does the chase algorithm terminate on all branches for a given $I$ and $\Sigma$?*

As presented, both these problems are undecidable in general. We also saw that the problem of testing if the core chase terminates for all input instances is undecidable in general. The undecidability result holds for the standard chase as well, in case we allow at least one denial constraint. Testing if the standard chase terminates for a given set of tgds on all instances remains however an open problem. Note that this problem is not the same as testing if there exists a universal solution for a given data-exchange setting with all input instances that is known to be an RE-complete problem [39].

Section 4 was dedicated to presenting large decidable classes of tgds for which it is known that the standard chase algorithm terminates on all branches for all input instances. As shown, all these classes actually ensure the termination for the "less expensive" (complexity based) semi-oblivious-chase algorithm, making this chase variation a better choice when dealing with sets of dependencies from those classes.

In case the chase based algorithm terminates, the instance computed is guaranteed to be homomorphic equivalent to any instance computed by any other chase variations. This property of the chase algorithms plays an important role in data exchange, especially in choosing the right instance on the target which should be materialized. Under the certain answers semantics for UCQ queries, the finite instances returned from any of the chase algorithms presented in Section 3 are good candidates to be materialized on the target. These instances, which are universal solutions, can be used together with the naïve evaluation to obtain the certain answers to any UCQ query over the target schema. In case a universal solution exists, the certain answers computation for UCQ queries is polynomial. In [12] it is shown that for some special classes of tgds, even if the universal solution is not guaranteed to exist, we may compute the certain answers to conjunctive queries. In these cases the complexity of computing the certain answers may grow as high as EXPTIME-hard.

To the best of our knowledge, the only chase based algorithm known to be complete in finding universal solutions for the data-exchange problem is the core chase. However, the core chase is the most expensive, complexity wise. This is because at each step it involves finding all the active triggers as well as computing the core of the produced instance. As shown in [20], the core-identification problem (i.e. given instances $I$ and $J$, *Is $I$ the core of $J$?*) is DP-complete. This leaves us with the open question if there exist other, less complex, chase based algorithms which are complete in finding universal solutions.

In this chapter we only focused on the cases where the chase algorithms terminate. This is mainly because in data exchange the infinite chase is not so important. If one is interested in the infinite chase, a good starting point would be [12].

## References

**1** Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

**2** Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41, 2009.

**3** A. V. Aho, C. Beeri, and J. D. Ullman. The theory of joins in relational databases. *ACM Trans. Database Syst.*, 4(3):297–314, 1979.

**4** Marcelo Arenas, Pablo Barceló, Ronald Fagin, and Leonid Libkin. Locally consistent transformations and query answering in data exchange. In *PODS*, pages 229–240, 2004.

**5** Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

**6** Marcelo Arenas, Ronald Fagin, and Alan Nash. Composition with target constraints. In *ICDT*, pages 129–142, 2010.

**7** Marcelo Arenas, Jorge Pérez, and Juan L. Reutter. Data exchange beyond complete data. In *PODS*, pages 83–94, 2011.

**8** Marcelo Arenas, Jorge Pérez, and Cristian Riveros. The recovery of a schema mapping: bringing exchanged data back. In *PODS*, pages 13–22, 2008.

**9** Renée J. Miller Ariel Fuxman, Phokion G. Kolaitis and Wang Chiew Tan. Peer data exchange. In *PODS*, pages 160–171, 2005.

**10** Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.

**11** Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. *Inf. Syst.*, 29(2):147–163, 2004.

**12** Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *KR*, pages 70–80, 2008.

**13** Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *ICDT*, pages 14–30, 2009.

**14** Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *PODS*, pages 77–86, 2009.

**15** Alin Deutsch. Fol modeling of integrity constraints (dependencies). In *Encyclopedia of Database Systems*, pages 1155–1161, 2009.

**16** Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

17 Alin Deutsch, Lucian Popa, and Val Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.

18 Ronald Fagin. Inverting schema mappings. In *PODS*, pages 50–59, 2006.

19 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.

20 Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.

21 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. In *PODS*, pages 83–94, 2004.

22 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Quasi-inverses of schema mappings. In *PODS*, pages 123–132, 2007.

23 Gösta Grahne and Adrian Onet. Data correspondence, exchange and repair. In *ICDT*, pages 219–230, 2010.

24 Gösta Grahne and Adrian Onet. Closed world chasing. In *LID*, pages 7–14, 2011.

25 Gösta Grahne and Adrian Onet. On conditional chase termination. In *AMW*, 2011.

26 Gösta Grahne and Adrian Onet. Anatomy of the chase. In *to appear*, 2013.

27 Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. Stratification criteria and rewriting techniques for checking chase termination. *PVLDB*, 4(11):1158–1168, 2011.

28 André Hernich. Answering non-monotonic queries in relational data exchange. In *ICDT*, pages 143–154, 2010.

29 André Hernich. Computing universal models under guarded tgds. In *ICDT*, pages 222–235, 2012.

30 André Hernich and Nicole Schweikardt. Cwa-solutions for data exchange settings with target dependencies. In *PODS*, pages 113–122, 2007.

31 David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

32 Phokion G. Kolaitis, Jonathan Panttaja, and Wang Chiew Tan. The complexity of data exchange. In *PODS*, pages 30–39, 2006.

33 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

34 Leonid Libkin. Data exchange and incomplete information. In *PODS*, pages 60–69, 2006.

35 Leonid Libkin. Incomplete information and certain answers in general data models. In *PODS*, pages 59–70, 2011.

36 David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

37 David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies (abstract). In *SIGMOD Conference*, page 152, 1979.

38 Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.

39 Bruno Marnette. *Tractable Schema Mappings Under Oblivious Termination.* PhD thesis, University of Oxford, 2010.

40 Bruno Marnette and Floris Geerts. Static analysis of schema-mappings ensuring oblivious termination. In *ICDT*, pages 183–195, 2010.

41 Michael Meier, Michael Schmidt, and Georg Lausen. On chase termination beyond stratification. *PVLDB*, 2(1):970–981, 2009.

42 Michael Meier, Michael Schmidt, Fang Wei, and Georg Lausen. Semantic query optimization in the presence of types. In *PODS*, pages 111–122, 2010.

43 Alberto O. Mendelzon. Database states and their tableaux. In *XP2 Workshop on Relational Database Theory*, 1981.

44 Adrian Onet. *The chase procedure and its applications.* PhD thesis, Concordia University, 2012.

45 Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

46 V Rutenberg. Complexity of generalized graph coloring. In *Proceedings of the 12th symposium on Mathematical foundations of computer science 1986*, pages 537–581, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

47 Francesca Spezzano and Sergio Greco. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.

48 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

49 Moshe Y. Vardi. Inferring multivalued dependencies from functional and join dependencies. *Acta Inf.*, 19:305–324, 1983.