Leveraging Equality Generating Dependencies for Chase Termination

Leveraging Equality Generating Dependencies for Chase Termination

Marco Calautti, Sergio Greco, Cristian Molinaro, and Irina Trubitsyna

DIMES, University of Calabria
87036 Arcavacata di Rende (CS) - Italy
{calautti, greco, cmolinaro, trubitsyna}@dimes.unical.it

Discussion Paper

Abstract. Tuple- and equality-generating dependencies have a wide range of applications in knowledge representation and databases, including ontological reasoning, data exchange, and data cleaning. In such settings, the chase is a central tool for many reasoning tasks. Since the chase evaluation might not terminate and it is undecidable whether it terminates, several termination criteria have been proposed, that is, (decidable) sufficient conditions ensuring termination. One of the main weaknesses of current approaches is the limited analysis they perform on equality-generating dependencies (EGDs).
In this paper, we show that an explicit analysis of EGDs can yield significant benefits and discuss a novel approach along this line.

1 Introduction

Tuple- and equality-generating dependencies (TGDs and EGDs, for short) were introduced for the purpose of database normalization and design, and have seen a revival of interest in the last years, with a wide range of applications in knowledge representation and databases (e.g., ontological reasoning, data exchange, and data cleaning). TGDs (a.k.a. existential rules) and EGDs are the basis of several prominent knowledge representation formalisms, such as Datalog+/– [14], and are closely related to the Horn fragments of the OWL 2 ontology language, as well as to Datalog with function symbols (e.g., see [26, 9, 12, 29, 28, 8, 13, 11, 30]).

In this regard, the chase [5] is a central tool for solving many reasoning tasks. It was originally proposed for classical database problems, such as query optimization, query containment and equivalence, dependency implication, and database schema design [1, 4, 27, 35]. In recent years, it has seen a revival of interest because of a wide range of applications where it plays a central role, such as data exchange, data cleaning and repairing, data integration, and ontological reasoning [19, 6, 3, 2, 15, 16, 21, 20, 25, 31].

The execution of the chase involves inserting tuples possibly with null values to satisfy tuple-generating dependencies, and replacing null values with constants or other null values to satisfy equality-generating dependencies. Specifically, the chase consists of applying a sequence of steps, where each step enforces a dependency that is not satisfied by the current instance. It might well be the case

that multiple dependencies can be enforced and, in this case, the chase picks one nondeterministically. Different choices lead to different sequences, some of which might be terminating, while others might not. This aspect is illustrated in the following example.

Example 1. Consider the set of dependencies Σ_1 below:

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

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

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

The importance of the chase in many applications is due to the fact that several problems (e.g., checking query containment under dependencies, checking implication of dependencies, computing solutions in data exchange, and computing certain answers in data integration) can be solved by exhibiting a *universal model*, and the chase computes a universal model, when it terminates [17]. Roughy speaking, a *model* for a database and a set of dependencies is a finite instance that includes the database and satisfies the dependencies. A *universal model* is a model that can be "mapped" to every other model—in a sense, it represents the entire space of possible models. Universal models are slight generalizations of universal solutions in data exchange [19], and can be used to compute them. Moreover, the certain answers to a conjunctive query in the presence of dependencies can be computed by evaluating the query over a universal model (rather than considering all models). Answering conjunctive queries over a set of facts extended with existential rules is a prominent problem in ontological reasoning. Other applications of universal models (e.g., dependency implication and query containment under dependencies) can be found in [17].

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

Indeed, as illustrated in Example 1 above, when we talk about chase termination, it is important to distinguish between two problems: checking whether

all chase sequences are terminating, and checking whether there is *at least one* terminating chase sequence. Most of the work in the literature has focused on the problem of checking if all chase sequences are terminating, independently of the considered database. However, since in many applications the ultimate goal is to compute a universal model, checking for the existence of a terminating chase sequence and constructing it suffices for the purpose.

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

Despite the significant body of work in this area, there are still large classes of dependencies for which the chase is not applicable as termination cannot be statically established. One weakness of current approaches is that the analysis of EGDs is limited or absent. In fact, more general approaches, such as *super-weak acyclicity* [36], semi-dynamic approaches *MFA* and *MSA* [24], and rewriting approaches [32–34], were meant to guarantee termination of TGDs only. Other approaches, such as *weak acyclicity* [19] and *safety* [37], guarantee the termination of a set of TGDs and EGDs, but do not analyze EGDs, which leads them to impose strong conditions on TGDs. Firing relations among dependences used in stratification-based approaches [17, 37, 33, 34] consider EGDs in a limited way. To mitigate the aforementioned issues, an "indirect" way of dealing with EGDs was proposed in [23, 36], where a set Σ of TGDs and EGDs is rewritten into a set Σ' containing only TGDs, and termination analysis is carried out on Σ'. The aim is to "simulate" the behavior of the EGDs by means of TGDs. While these preprocessing steps ensure soundness, i.e., if all chase sequences of Σ' are terminating then all chase sequences of Σ are terminating, they are not complete, i.e., the implication in the opposite direction does not hold.

Treating EGDs as first-class citizens is very important, as they are among the most popular classes of dependencies in real applications, playing a critical role in maintaining data integrity, query optimization and indexing, and schema design [18]. For instance, functional dependencies can be expressed by EGDs. In very simple scenarios, such as Example 1 above, current termination criteria are not able to say whether a universal model can be found. As a further scenario, Example 2 shows a simple set of dependencies for which all chase sequences are terminating, but there is no terminating chase sequence for the set of dependencies obtained from the EGD simulation.

In this paper, we show that an explicit analysis of EGDs can yield significant benefits, and discuss a novel approach able to perform this kind of analysis.

2 Dealing with EGDs

In this section, we discuss several issues that arise in the presence of EGDs.

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

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

We will denote by $\mathsf{CT}^{\mathsf{c}}_{\forall}$ (resp. $\mathsf{CT}^{\mathsf{c}}_{\exists}$), with $\mathsf{c} \in \{\mathsf{std}, \mathsf{obl}, \mathsf{sobl}, \mathsf{core}\}$, the class of sets of dependencies Σ such that for every database D all (resp. some) c-chase sequences of D with Σ are terminating. Here, std, obl, sobl and core refer to different variants of the chase, namely *standard*, *oblivious*, *semi-oblivious*, and *core*—we refer the reader to [10, 7] for more details on them. In the rest of the paper, given two sets C_1 and C_2, we write $C_1 \nparallel C_2$ iff $C_1 \not\subseteq C_2$ and $C_2 \not\subseteq C_1$.

The relationships between the aforementioned classes have been studied in the presence of TGDs only in [38]. Specifically, it has been shown that:

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

In the presence of arbitrary sets of dependencies (possibly including both TGDs and EGDs), the relationships between the classes $\mathsf{CT}^{\mathsf{c}}_{\mathsf{q}}$, where $\mathsf{q} \in \{\forall, \exists\}$ and $\mathsf{c} \in \{\mathsf{std}, \mathsf{obl}, \mathsf{sobl}, \mathsf{core}\}$, become those in Table 1, which reports the relationships for the case of TGDs only and when both TGDs and EGDs are allowed.

As already mentioned, chase termination criteria proposed in the literature focus on TGDs considering EGDs in a very limited way. An "indirect" way of dealing with EGDs has been proposed in [23, 36]. Specifically, the analysis of a set of dependencies Σ containing both TGDs and EGDs is performed on a set Σ' derived from Σ and containing only TGDs. The aim is to "simulate" the behavior of the EGDs by means of TGDs only. The first approach of this kind, known as *natural simulation*, has been proposed in [23], and further refined by the *substitution-free simulation* in [36]. Below is an example showing how the substitution-free simulation works.

Example 2. Consider the following set of dependencies Σ_2:

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

The substitution-free simulation works as follows:

1. *The TGDs below (equality-axioms) are added to Σ_2:*

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

2. *Every occurrence of $x = y$ in Σ_2 is replaced with $Eq(x,y)$. In our case, this affects r_4 and r_5 only, which are replaced with:*

$$r'_4: A(x) \wedge A(y) \rightarrow Eq(x,y) \qquad r'_5: B(x) \wedge B(y) \rightarrow Eq(x,y)$$

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

$$r_1' : A(x_2) \wedge B(x) \wedge Eq(x, x_2) \rightarrow C(x)$$
$$r_1'' : A(x) \wedge B(x_2) \wedge Eq(x, x_2) \rightarrow C(x)$$

Notice that the only dependencies that remain unchanged are r_2 and r_3 and that no EGDs occur in the resulting set of dependencies.

Although somehow left implicit in [23, 36], the natural simulation and the substitution-free simulation ensure the desirable *soundness* property: if $\Sigma' \in \mathsf{CT}^{\mathsf{c}}_{\forall}$, then $\Sigma \in \mathsf{CT}^{\mathsf{c}}_{\forall}$, for $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$. The natural question now is whether these simulations are also *complete*, that is, if the implication in the opposite direction holds. The answer is negative for both approaches.

For example, the set of dependencies Σ_2 of Example 2 above belongs to $\mathsf{CT}^{\mathsf{c}}_{\forall}$ (and thus belongs to $\mathsf{CT}^{\mathsf{c}}_{\exists}$ too), but any of its substitution-free simulations is not even in $\mathsf{CT}^{\mathsf{c}}_{\exists}$, for every $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$. The problem is that the simulation of EGDs by means of TGDs is not able to fully capture the specific behavior of EGDs, which replace null values (with constants and other null values). This aspect is not faithfully modeled by storing the information that a null value is equal to a constant or to another null value.

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

Example 3. Consider the set of dependencies Σ_1 of Example 1 and the database $D = \{N(a)\}$. There is no terminating c-chase sequence of D_1 with the set of TGDs $\Sigma_1' = \{r_1, r_2\}$, for every $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$. In fact, it is easy to see that an infinite number of facts is introduced: $E(a, \eta_1)$, $N(\eta_1)$, $E(\eta_1, \eta_2), \ldots$. However, the addition of the EGD r_3 allows us to have a terminating c-chase sequence, obtained by enforcing first r_1 and then r_3, and whose result is the universal model $\{N(a), E(a, a)\}$.

Example 4. Consider the set of dependencies Σ_4 below:

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

For every database D, every c-chase sequence of D with the set of TGDs $\Sigma_4' = \{r_1, r_2\}$ is terminating, for every $\mathsf{c} \in \{\mathsf{obl}, \mathsf{sobl}, \mathsf{std}\}$. On the other hand, there is no terminating c-chase sequence of $D = \{N(a)\}$ with Σ_4, as an infinite number of facts is introduced: $E(a, \eta_1, \eta_1)$, $N(\eta_1), E(\eta_1, \eta_2, \eta_2)$, $N(\eta_2), \ldots$.

3 A Rewriting Approach

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

Thus, the input is a set of dependencies Σ, while the output is a set of adorned dependencies Σ^{μ} along with a boolean value *Acyc*. More specifically, if *Acyc* is *false*, then a form of cyclicity has been detected; otherwise, for every database D, there is a terminating standard chase sequence of D with Σ. As for the second aim of the algorithm, the adorned set of dependencies Σ^{μ} given as output can be used as follows: a sufficient condition for checking membership in $\mathsf{CT}^{\mathsf{std}}_{\exists}$ is applied to Σ^{μ} rather than Σ. If Σ^{μ} satisfies the condition, then the original set of dependencies Σ is in $\mathsf{CT}^{\mathsf{std}}_{\exists}$.

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

The algorithm makes use of *adornment definitions*, which are expressions of the form $f_i = f^r_z(\alpha)$, whose intuitive meaning is that z is an existentially quantified variable in the head of the TGD r and α is an adornment for the body of r. Roughly speaking, adornment definitions are some sort of provenance information used to keep track of the "history" of a null.

The following example gives the basic idea of how the technique works.

Example 5. Consider the set of dependencies Σ_1 of Example 1. Initially, the following two adorned dependencies, mapping unadorned atoms to atoms adorned with strings of b's, are added to Σ^{μ}_1:

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

The algorithm then proceeds by adorning full dependencies and adds the following adorned dependencies to Σ^{μ}_1:

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

Next, the existentially quantified dependency (namely, r_1) is adorned and the following adorned dependency is added to Σ^{μ}_1:

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

Moreover, the following adornment definition is derived $f_1 = f_y^{r_1}(b)$*. After that, the algorithm starts considering full dependencies again. By adorning the EGD* r_3*, the following adorned dependency is obtained, which is added to* Σ_1^μ*:*

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

The adorned EGD s_6 *is analyzed by the algorithm, which replaces* f_1 *with* b *in* Σ_1^μ*. The resulting set of adorned dependencies is* $\Sigma_1^\mu = \{s_1, s_2, s_3, s_4, s_5'\}$*, where* s_5' *is derived from* s_5 *by replacing* f_1 *with* b*, that is,* $s_5' : N^b(x) \rightarrow \exists y\ E^{bb}(x, y)$*.*

At this point, no further dependencies can be adorned and the algorithm terminates by returning the value $Acyc = true$ *along with* Σ_1^μ*.*

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. M. Arenas, P. Barceló, R. Fagin, and L. Libkin. Locally consistent transformations and query answering in data exchange. In *PODS*, pages 229–240, 2004.
3. M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.
4. C. Beeri and M. Y. Vardi. Formal systems for tuple and equality generating dependencies. *SIAM J. Comput.*, 13(1):76–98, 1984.
5. C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.
6. L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*, pages 268–279, 2011.
7. M. Calautti, G. Gottlob, and A. Pieris. Chase termination for guarded existential rules. In *PODS*, pages 91–103, 2015.
8. M. Calautti, S. Greco, C. Molinaro, and I. Trubitsyna. Checking termination of logic programs with function symbols through linear constraints. In *RuleML*, pages 97–111, 2014.
9. M. Calautti, S. Greco, C. Molinaro, and I. Trubitsyna. Logic program termination analysis using atom sizes. In *IJCAI*, pages 2833–2839, 2015.
10. M. Calautti, S. Greco, C. Molinaro, and I. Trubitsyna. Exploiting equality generating dependencies in checking chase termination. *PVLDB*, 9(5):396–407, 2016.
11. M. Calautti, S. Greco, C. Molinaro, and I. Trubitsyna. Using linear constraints for logic program termination analysis. *To appear in TPLP*, 2016.
12. M. Calautti, S. Greco, F. Spezzano, and I. Trubitsyna. Checking termination of bottom-up evaluation of logic programs with function symbols. *TPLP*, 15(6):854–889, 2015.
13. M. Calautti, S. Greco, and I. Trubitsyna. Detecting decidable classes of finitely ground logic programs with function symbols. In *PPDP*, pages 239–250, 2013.
14. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.
15. A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.
16. G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. On reconciling data exchange, data integration, and peer data management. In *PODS*, pages 133–142, 2007.

17. A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.
18. R. Fagin. Equality-generating dependencies. In *Encyclopedia of Database Systems*, pages 1009–1010. 2009.
19. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Th. Comp. Sc.*, 336(1):89–124, 2005.
20. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM TODS*, 33(2), 2008.
21. F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 6(9):625–636, 2013.
22. T. Gogacz and J. Marcinkowski. All-instances termination of chase is undecidable. In *ICALP*, pages 293–304, 2014.
23. G. Gottlob and A. Nash. Efficient core computation in data exchange. *J. ACM*, 55(2), 2008.
24. B. C. Grau, I. Horrocks, M. Krotzsch, C. Kupke, D. Magka, B. Motik, and Z. Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *JAIR*, 47:741–808, 2013.
25. G. Greco, S. Greco, I. Trubitsyna, and E. Zumpano. Optimization of bound disjunctive queries with constraints. *TPLP*, 5(6):713–745, 2005.
26. S. Greco and C. Molinaro. *Datalog and Logic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.
27. S. Greco, C. Molinaro, and F. Spezzano. *Incomplete Data and Data Dependencies in Relational Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
28. S. Greco, C. Molinaro, and I. Trubitsyna. Bounded programs: A new decidable class of logic programs with function symbols. In *IJCAI*, pages 926–932, 2013.
29. S. Greco, C. Molinaro, and I. Trubitsyna. Logic programming with function symbols: Checking termination of bottom-up evaluation through program adornments. *TPLP*, 13(4-5):737–752, 2013.
30. S. Greco, C. Molinaro, I. Trubitsyna, and E. Zumpano. NP datalog: A logic language for expressing search and optimization problems. *TPLP*, 10(2):125–166, 2010.
31. S. Greco, C. Sirangelo, I. Trubitsyna, and E. Zumpano. Feasibility conditions and preference criteria in querying and repairing inconsistent databases. In *DEXA*, pages 44–55, 2004.
32. S. Greco and F. Spezzano. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.
33. S. Greco, F. Spezzano, and I. Trubitsyna. Stratification criteria and rewriting techniques for checking chase termination. *PVLDB*, 4(11):1158–1168, 2011.
34. S. Greco, F. Spezzano, and I. Trubitsyna. Checking chase termination: Cyclicity analysis and rewriting techniques. *IEEE TKDE*, 27(3):621–635, 2015.
35. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM TODS*, 4(4):455–469, 1979.
36. B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.
37. M. Meier, M. Schmidt, and G. Lausen. On chase termination beyond stratification. *PVLDB*, 2(1):970–981, 2009.
38. A. Onet. The chase procedure and its applications in data exchange. In *Data Exchange, Integration, and Streams*, pages 1–37. 2013.