Rewriting-based Check of Chase Termination

Marco Calautti, Sergio Greco, Cristian Molinaro, and Irina Trubitsyna
{calautti,greco,cmolinaro,trubitsyna}@dimes.unical.it

DIMES, Università della Calabria, 87036 Rende (CS), Italy

Keywords: Chase, Data Dependencies, Data Exchange, Data Integration.

1 Introduction

The Chase is a fixpoint algorithm enforcing satisfaction of data dependencies (also called constraints) in databases. It has been proposed more than thirty years ago [2,18] and has seen a revival of interest in recent years in both database theory and practical applications. Indeed, the availability of data coming from different sources easily results in inconsistent or incomplete data (i.e., data not satisfying data dependencies) and, therefore, techniques for fixing inconsistencies are crucial [1,3,5,7,8,13,17].

The chase algorithm is used, directly or indirectly, on an everyday basis by people who design databases, and it is used in commercial systems to reason about the consistency and correctness of a data design. New applications of the chase in meta-data management, ontological reasoning, data exchange, data cleaning, and query optimization have been proposed as well [6,9].

The chase algorithm solves possible violations of constraints by inserting new tuples, possibly containing null values [4]. The following example shows a case where a given database does not satisfy a set of *tuple generating dependencies* (TGDs) and the application of the chase algorithm produces a new consistent database by adding tuples with nulls.

Example 1. Consider the following set of constraints Σ_1 describing departments and their employees:

$$\begin{array}{l}\forall x\,\forall y\ Department(x) \wedge Managed(x,y) \rightarrow Employee(y)\\ \forall x\ Employee(x) \rightarrow \exists\, y\ WorksFor(x,y)\\ \forall x\,\forall y\ WorksFor(x,y) \rightarrow \exists\, z\ Managed(y,z)\end{array}$$

Consider the database $D = \{Department(d), Managed(d,m)\}$. Since the first constraint is not satisfied, the tuple $Employee(m)$ is inserted. This update operation fires the second constraint to insert the tuple $WorksFor(m,\eta_1)$, which in turn fires the third constraint so that the tuple $Managed(\eta_1,\eta_2)$ is added to the database (η_1 and η_2 are new labeled nulls). At this point, the chase terminates since the database is consistent, that is, all dependencies are satisfied. □

Unfortunately, the chase algorithm may not terminate. For instance, in Example 1 if we delete from the first constraint the atom $Department(x)$, the chase never terminates and adds an infinite number of tuples to the database. It has been formally proved in [12] that the problem of deciding whether the chase procedure terminates is semi-decidable. The first and basic effort concerning the formalization of a (decidable) sufficient condition guaranteeing chase termination is *weak acyclicity* [11]. Informally, it checks whether the constraints do not allow for nulls to cyclically propagate inside predicates' positions. Considering the example above, we have that a value in the second position of predicate $Managed$ is copied to $Employee$ (denoted by $M_2 \rightarrow E_1$). This forces the introduction of a new null value in the second position of $WorksFor$, (denoted as $E_1 \xrightarrow{*} WF_2$); this value is then copied in the first position of $Managed$ ($WF_2 \rightarrow M_1$) and it also forces the introduction of a new null value in the second position of $Managed$ ($WF_2 \xrightarrow{*} M_2$). Since it is possible to reach position M_2 from itself through a connection of the form $\xrightarrow{*}$, an infinite number of nulls could be introduced during the chase procedure.

Different extensions of *weak acyclicity* have been proposed. *Safety* [20] and *super-weak acyclicity* [19] identify the positions in which null values can be propagated. *Stratification*-based approaches [10,20,16] analyse whether dependencies may fire each other and thus propagate null values from one to another. See [14] for a comprehensive survey on this topic. Nevertheless, despite the previously mentioned results, there are still important classes of terminating data dependencies which are not identified by any of the previously mentioned criteria: Example 1 showed one such a case. To overcome such limitations, rewriting techniques have been proposed [15,16]. In the following section we give an overview on them and show how the constraints of Example 1 can be rewritten by using predicate adornments in order to allow simple termination conditions (even the simplest one, weak-acyclicity) to understand that the chase procedure terminates. Issues regarding the extension of these techniques to managing also *equality generating dependencies (EGDs)* are discussed in Section 3.

2 Constraint Rewriting

We start by introducing the basic idea of the *Adn* technique [15], which can be used in conjunction with current termination criteria, enabling us to detect more sets of constraints as terminating. The technique consists of rewriting a set of TGDs Σ into a new set Σ^α which is "better" than the original one for the purpose of checking termination. Rather than applying a termination criterion to Σ, the new set Σ^α is used and if Σ^α satisfies the criterion then chase termination for Σ is guaranteed. This allows us to recognize larger classes of constraints for which chase termination is guaranteed: if Σ satisfies chase termination criterion C, then the rewritten set Σ^α satisfies C as well, but the vice versa is not true, that is, there are significant classes of constraints for which Σ^α satisfies C and Σ does not.

Example 2. Consider again the set of TGDs Σ_1. The *Adn* technique first rewrites TGDs by associating strings of b symbols to body atoms and to head positions containing universally quantified variables. Then, f symbols are associated for existentially quantified variables. This new set of TGDs is denoted by $Base(\Sigma_1)$:

$$\begin{array}{l}
\forall x\,\forall y\ Department^{b}(x) \wedge Managed^{bb}(x,y) \rightarrow Employee^{b}(y) \\
\forall x\ Employee^{b}(x) \rightarrow \exists\, y\ WorksFor^{bf}(x,y) \\
\forall x\,\forall y\ WorksFor^{bb}(x,y) \rightarrow \exists\, z\ Managed^{bf}(y,z)
\end{array}$$

Subsequently, because of the presence of atoms $WorksFor^{bf}(x,y)$ and $Managed^{bf}(x,y)$ in $Base(\Sigma_1)$, the rewriting continues by producing the following set of TGDs $Derived(\Sigma_1)$:

$$\begin{array}{l}
\forall x\,\forall y\ WorksFor^{bf}(x,y) \rightarrow \exists\, z\ Managed^{ff}(y,z) \\
\forall x\,\forall y\ Department^{b}(x) \wedge Managed^{bf}(x,y) \rightarrow Employee^{f}(y) \\
\forall x\ Employee^{f}(x) \rightarrow \exists\, y\ WorksFor^{ff}(x,y) \\
\forall x\,\forall y\ WorksFor^{ff}(x,y) \rightarrow \exists\, z\ Managed^{ff}(y,z)
\end{array}$$

At this point, the generation of $Derived(\Sigma_1)$ terminates, since the atom $Department^{b}(x)$ cannot be joined with $Managed^{ff}(x,y)$ to produce a new adorned TGD. The rewritten set of TGDs $Adn(\Sigma_1)$ is weakly-acyclic, whereas the original set Σ_1 is not recognized by any chase termination criteria. $\square$

Rewriting Algorithm Improvement. The rewriting algorithm *Adn* has been further improved into the Adn^{+} algorithm [16] by using different adornments for each existentially quantified variable and by considering how TGDs may fire each other in the generation of adorned atoms. During the rewriting process, this algorithm also performs a basic cyclicity check, allowing to eventually determine the termination of the chase, without necessarily relying on other criteria. The new criterion is called *Acyclicity*. To the best of our knowledge, the class of TGDs recognized by this criterion is the most general class known so far.

3 Adding EGDs

In the previous sections, we have considered the case where all constraints are TGDs. In this section, we show how the chase termination problem radically changes when we allow also EGDs.

Given a set of TGDs Σ for which the chase does not terminate, we can show that the addition of EGDs to Σ may allow to have a terminating chase sequence. On the other hand, if the chase always terminates for Σ, adding EGDs to Σ may make the chase of Σ non-terminating.

Example 3. Consider the following two sets of constraints Σ_3 (left) and Σ_3' (right):

$$\begin{array}{ll}
r_1:\ \forall x\ A(x) \rightarrow \exists y\ N(y) & r_1':\ \forall x\ N(x) \rightarrow \exists y\,\exists z\, S(x,y,z) \\
r_2:\ \forall x\ N(x) \rightarrow \exists y\ E(x,y) & r_2':\ \forall x\,\forall y\,\forall z\ S(x,y,y) \rightarrow N(y) \\
r_3:\ \forall x\,\forall y\ E(x,y) \rightarrow N(y) & r_3':\ \forall x\,\forall y\,\forall z\ S(x,y,z) \rightarrow T(x,y,z) \\
r_4:\ \forall x\,\forall y\ E(x,y) \rightarrow x = y & r_4':\ \forall x\,\forall y\,\forall z\ T(x,y,z) \rightarrow y = z
\end{array}$$

and the database $D = \{N(a)\}$. The chase applied to the database D and the subset of TGDs $\{r_1, r_2\}$ of Σ_3 is not terminating as it introduces an infinite number *of tuples $E(\eta_1, \eta 2), E(\eta_3, \eta_1), ...$ The introduction of the EGD r_3 allows to have a terminating sequence, which produces the universal solution $\{N(a), E(a, a)\}$.*

The subset of TGDs $\{r'_1, r'_2, r'_3\}$ of Σ'_3 is terminating for all database instances as recognized by several criteria (e.g., super-weak acyclicity). However, the chase fixpoint applied to Σ'_3 and the database D is non-terminating as it introduces an infinite number of tuples $S(a, \eta_1, \eta_1), T(a, \eta_1, \eta_1), N(\eta_1),\ S(\eta_1,\ \eta_2, \eta_2), T(\eta_1, \eta_2, \eta_2), N(\eta_2), ...$ □

As shown in the previous example, when for a set of dependencies it is not the case that every chase sequence is terminating, the existence of at least one terminating chase sequence, for every database, might still be guaranteed. Thus, one could extend rewriting techniques such as Adn^+ to sets of TGDs and EGDs, in order to find whether there exists, for every database, at least one terminating chase sequence. In order to cope with the aforementioned issues, algorithm Adn^+ can be extended in such a way that some adornments generated by rewriting TGDs are changed in order to satisfy the head equalities of EGDs. Specifically, the algorithm first tries to adorn as many EGDs as possible, and then consider the rewriting of a single TGD. The basic idea is illustrated in the following example.

Example 4. Consider the set of dependencies Σ_3 of Example 3. As initially EGD r_4 cannot be adorned, TGD r_1 is rewritten into:

$$\forall x\ A^b(x) \to \exists y\ N^{f_1}(y)$$

and r_2 is rewritten into:

$$\forall x\ N^b(x) \to \exists y\ E^{bf_2}(x, y)$$

Now, EGD r_4 can be used to "merge" distinct symbols. This is accomplished by constructing the following adorned version of r_4 using the atom $E^{bf_2}(x, y)$:

$$\forall x\, \forall y\ E^{bf_2}(x, y) \to x = y$$

This indicates that every occurrence of the symbol f_2 in the obtained adorned dependencies has to be replaced with b, thereby obtaining:

$$\begin{array}{l}\forall x\ A^b(x) \to \exists y\ N^{f_1}(y)\\ \forall x\, \forall y\ N^b(x) \to \exists y\ E^{bb}(x, y)\\ \forall x\, \forall y\ E^{bb}(x, y) \to x = y\end{array}$$

Then, TGD r_3 is adorned, obtaining:

$$\forall x\, \forall y\ E^{bb}(x, y) \to N^b(y)$$

Then, TGD r_2 is adorned using atom $N^{f_1}(x)$, obtaining:

$$\forall x\, \forall y\ N^{f_1}(x) \to \exists y\ E^{f_1 f_3}(x, y)$$

Again, EGD r_4 is used, getting:

$$\forall x\, \forall y\ E^{f_1 f_3}(x,y) \rightarrow x = y$$

Consequently, f_3 is replaced with f_1 and we get:

$$\begin{array}{l}
\forall x\ A^b(x) \rightarrow \exists y\ N^{f_1}(y) \\
\forall x\, \forall y\ N^b(x) \rightarrow \exists y\ E^{bb}(x,y) \\
\forall x\, \forall y\ E^{bb}(x,y) \rightarrow N^b(y) \\
\forall x\, \forall y\ N^{f_1}(x) \rightarrow \exists y\ E^{f_1 f_1}(x,y) \\
\forall x\, \forall y\ E^{bb}(x,y) \rightarrow x = y \\
\forall x\, \forall y\ E^{f_1 f_1}(x,y) \rightarrow x = y
\end{array}$$

Finally, atom $E^{f_1 f_1}(x,y)$ is used to adorn r_3, obtaining:

$$\forall x\, \forall y\ E^{f_1 f_1}(x,y) \rightarrow N^{f_1}(y)$$

At this point, the rewriting stops, since no new adorned dependency can be constructed. Intuitively, the algorithm identifies the existence of a terminating chase sequence because symbols f_1, f_2, f_3, which represent nulls constructed w.r.t. the symbols occurring in the body of the TGD, are not "cyclic" in the following sense. Symbol f_1 "depends on" symbol b in $body(r_1)$, f_2 depends on symbol b in $body(r_2)$, and f_3 depends on symbol f_1. Since no pair of symbols f_i, f_j in the final set of dependency is such that f_i depends on f_j and vice versa, the set Σ_3 has a terminating chase sequence. □

References

1. F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41, 2009.
2. 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.
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. L. E. Bertossi. Consistent query answering in databases. *SIGMOD Record*, 35(2):68–76, 2006.
6. A. Cali, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.
7. L. Caroprese, S. Greco, and E. Zumpano. Active integrity constraints for database consistency maintenance. *IEEE Trans. Knowl. Data Eng.*, 21(7):1042–1058, 2009.
8. J. Chomicki. Consistent query answering: Five easy pieces. In *ICDT*, pages 1–17, 2007.
9. 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.
10. A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

11. 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.
12. T. Gogacz and J. Marcinkowski. All-instances termination of chase is undecidable. In *ICALP*, pages 293–304, 2014.
13. G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *TKDE*, 15(6):1389–1408, 2003.
14. S. Greco, C. Molinaro, and F. Spezzano. *Incomplete Data and Data Dependencies in Relational Databases.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
15. S. Greco and F. Spezzano. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.
16. S. Greco, F. Spezzano, and I. Trubitsyna. Stratification criteria and rewriting techniques for checking chase termination. *PVLDB*, 4(11):1158–1168, 2011.
17. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
18. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.
19. B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.
20. M. Meier, M. Schmidt, and G. Lausen. On chase termination beyond stratification. *CoRR*, abs/0906.4228, 2009.