

On using simplification and correction tables for integrity maintenance in integrated databases

Henning Christiansen

Roskilde University
Computer Science Dept.
P.O.Box 260, DK-4000 Roskilde, Denmark
henning@ruc.dk

Davide Martinenghi

Free University of Bozen/Bolzano
Faculty of Computer Science
P.zza Domenicani, 3, I-39100 Bolzano, Italy
martinenghi@inf.unibz.it

Abstract

When a database is defined as views over autonomous sources, inconsistencies with respect to global integrity constraints are to be expected. This paper investigates the possibility of using a technique known as simplification of integrity constraints in order to maintain, in an incremental way, a correction table of virtual updates which, if executed, would restore consistency. Access to global data is then made through auxiliary views that take the table into account. This structure is relevant for a service that draws upon external data sources over which it has no write permission. The approach employs assumptions about local source consistency as well as cross-source hypotheses whenever possible.

1 Introduction

We study the problem of maintaining consistency of global integrity constraints in a data integration (DI) system, i.e., a system providing a reconciled and unified view of a set of possibly distributed and heterogeneous data sources. We refer to the approach known as global-as-view (GaV). Our basic assumption is that sources are autonomous in the sense that the integrating systems, henceforth referred to as the *global* level, cannot update sources; the global level can ask queries to and receive answers from each source, and we assume, furthermore, that each source maintains its own local integrity constraints and provides information about the updates that it has received.

Under the assumption that all integrity constraints (ICs) at global as well as local levels are correct properties of the world, a violation of a global IC means that at least one of the sources has wrong (or lacking) information. To maintain global consistency, it seems natural to introduce a correction table of virtual updates which, if executed, would restore

consistency; access to global data can then be made through auxiliary views that take the table into account (transformation of global queries into a set of source queries is straightforward under GaV and thus not considered in this paper).

Simplification of ICs is a principle introduced in the early 1980's for single databases by [9] and which means to optimize IC checking based on the assumption that the database was consistent before the update. This implies an incremental checking and maintenance of ICs so that (ideally) only a minimal number of tuples potentially affected by each update are checked. In the version of [9] and several following works, the check is performed after the update so that problematic updates need to be undone or corrected. This has been improved in later work by checking before each proposed update so that illegal updates are not even executed. Such updates can be parameterized so that the potentially complex symbolic transformations involved can be performed at database design time with actual parameters plugged into test queries at runtime. The principle has been applied to detect inconsistency in DI by [5] and in the present work we extend this with a correction table which makes it possible to maintain a virtual view of consistency. The use of such a table of virtual corrections has been described by [7] as a way to treat “what-if” queries and to make undoing easy, but the combination with simplification seems new. Other approaches to consistency maintenance in DI, e.g., [3, 4, 12], modify source relations and do not incrementally trace changes. Instead of maintaining (virtual) consistency, approaches to consistent query-answering [1, 2] return only tuples that the different databases agree on. A recent survey on simplification methods can be found in [8].

In the following we introduce correction tables and consider their application for consistency maintenance using simplification. Proofs are omitted due to space constraints.

2 Databases and data integration systems

To integrate a set of different local data sources means to provide a common database schema (the global schema), and describe a relationship between the different local schemata and the global one. Two common paradigms for defining such relationships are the so-called *local-as-view* (LaV) and *global-as-view* (GaV); see [11] for definitions and comparison. We refer in this paper to the GaV approach, in which global relations are defined as views over local ones; we also assume that the mappings are *sound*, i.e., the information produced by the views over the sources contains only, but not necessarily all the data associated to the global predicates. We adopt DATALOG with default negation [10] and its terminology as the formalism to express the different components of a database. We refer to *clauses*, including *facts* (ground clauses whose body is empty, understood as *true*) for the tuples of a relation, denials (clauses whose head is empty, understood as *false*) for integrity constraints, and *rules* (all other clauses) to define views. Built-in predicates including $=$ and \neq are assumed with their standard meaning; non-built-in predicates are called base predicates. All clauses are assumed to be *range restricted* meaning that any variable in a clause appears (also) in some positive database literal in the body of that clause.

Only nonrecursive databases are considered, so a standard, least Herbrand model semantics can be used; $D \models \phi$ means that a formula ϕ holds in the standard model of database D .

Definition 2.1 A source database $S = \langle \Pi, F, \Gamma \rangle$ consists of a set of predicates Π , a set of facts F (the state) over Π plus a constraint theory Γ (set of denials over Π). A source S is consistent whenever $F \models \Gamma$, inconsistent otherwise.

In the following, we assume that the set of predicates is understood and simply write $S = \langle F, \Gamma \rangle$ to indicate a source.

A DI system consists of a collection of source databases with pairwise disjoint predicate sets, a set of predicates (called *global*) which is disjoint with any of the source databases and defined as views over the source predicates. Furthermore, it has a set of *cross-source hypotheses* which are denials that involve predicates from different sources; as an example, we may have that the domain of two attributes in different sources are disjoint. Finally, there are global integrity constraints, which are denials involving global predicates only. Again, we assume that the sets of source as well as global predicates are understood.

Definition 2.2 A DI system D is a quadruple $\langle F, \Delta, W, \Gamma \rangle$, where F is the set of all source facts, Δ the set of trusted hypotheses consisting of all source integrity constraints and cross-source hypotheses such that $F \models \Delta$, W the view definitions for the global predicates over source predicates, and

Γ a set of global integrity constraints. D is consistent whenever $F \cup W \models \Gamma$, inconsistent otherwise.

Without giving full details, we assume that in the global integrity constraints each view predicate can be replaced by its definition in a truth-preserving way, referred to as *unfolding*, such that the result is a set of range restricted denials in which only source or built-in predicates occur¹. Under this assumption, the W component can be omitted and we will simply indicate $D = \langle F, \Delta, \Gamma \rangle$, where Γ are the global integrity constraints after unfolding, and thus expressed in terms of source predicates. We may use $D \models \Gamma$ as alternative notation for $F \models \Gamma$ and similarly $D \cup X \models \Gamma$ for $F \cup X \models \Gamma$ (X an arbitrary set of formulas).

Definition 2.3 A constraint theory is consistently signed if no base predicate occurs in both a positive and a negative literal. A DI system $\langle F, \Delta, \Gamma \rangle$ is consistently signed whenever $\Delta \cup \Gamma$ is consistently signed.

Example 2.1 Consider two sources, $S_1 = \langle F_1, \Gamma_1 \rangle$ and $S_2 = \langle F_2, \Gamma_2 \rangle$ with predicates m_1 and m_2 and a mediator G with global predicate m_0 (husband, wife) describing marriages and defined as $m_0(X, Y) \leftarrow m_1(X, Y) \vee m_2(X, Y)$. On all databases an integrity constraint is imposed enforcing non-bigamism of husbands: $\Gamma_i = \{ \leftarrow m_i(X, Y) \wedge m_i(X, Z) \wedge Y \neq Z \}$, for $i = 0, 1, 2$. The unfolding of Γ_0 with respect to the view (each disjunction causes denial splitting) is

$$\Gamma = \{ \begin{array}{l} \leftarrow m_1(X, Y) \wedge m_1(X, Z) \wedge Y \neq Z, \\ \leftarrow m_1(X, Y) \wedge m_2(X, Z) \wedge Y \neq Z, \\ \leftarrow m_2(X, Y) \wedge m_1(X, Z) \wedge Y \neq Z, \\ \leftarrow m_2(X, Y) \wedge m_2(X, Z) \wedge Y \neq Z \end{array} \}$$

The DI system $G = \langle F_1 \cup F_2, \Gamma_1 \cup \Gamma_2, \Gamma \rangle$ is consistently signed.

We disregard the problem of updating sources and maintaining integrity locally; we simply assume that each local database only allows consistency preserving updates (checked locally), and that the global database is informed about such updates in terms of which tuples are actually added and deleted. In other words, an update U exists only in the context of some database and, if U expresses the addition of some atom A , this implies that A was not in the database before the update; similarly for deletions. An update is indicated by a set of ground literals: the negative ones indicate deletion, the positive ones addition. For any atom A , we indicate with $neg(A)$ the literal $\neg A$, and with $neg(\neg A)$ the literal A . Application and composition of updates are defined using the following notation.

¹As discussed, e.g., in [6], this is only possible if views are nonrecursive, as we assume here, and if view predicates whose definitions make projections (i.e., contain non-distinguished variables) are negated an even number of times. If the latter condition is not satisfied, one should allow conjuncts including $\neg\exists$ constructs to occur in denials.

Definition 2.4 Let U and V be sets of literals; the composition $U \circ V$ is defined as $(U \cup V) \setminus \{L \mid \{L, \text{neg}(L)\} \subseteq U \cup V\}$.

For an update U , we use the notation $\neg U$ to refer to the set of literals $\{\text{neg}(L) \mid L \in U\}$.

Definition 2.5 Let $D = \langle F, \Delta, \Gamma \rangle$ be a DI system. An update U (for D) is a set of ground source literals such that $A \in U$ implies $A \not\models F$. The updated DI system is defined as $D^U = \langle F \circ U, \Delta, \Gamma \rangle$.

3 Simplification

Here we briefly review the framework of [6] for simplification of ICs. The following operator **After** transforms a constraint theory Γ into a constraint theory Σ that evaluates in any state D in the same way Γ would evaluate in an updated state D^U .

Definition 3.1 Let Γ be a constraint theory and U an update; $\text{After}^U(\Gamma)$ refers to a copy of Γ in which all atoms of the form $p(\vec{t})$ have been simultaneously replaced by $(p(\vec{t}) \vee \vec{t} = \vec{a}_{p,1} \vee \dots \vee \vec{t} = \vec{a}_{p,n_p}) \wedge \vec{t} \neq \vec{b}_{p,1} \wedge \dots \wedge \vec{t} \neq \vec{b}_{p,m_p}$, where $\{p(\vec{a}_{p,1}), \dots, p(\vec{a}_{p,n_p}), \neg p(\vec{b}_{p,1}), \dots, \neg p(\vec{b}_{p,m_p})\}$ is the biggest subset of U containing only p -literals; $\vec{a}_{i,j}$ and $\vec{b}_{i,j}$ are sequences of constants, \vec{t} a sequence of terms.

We also assume that **After** performs trivial truth preserving transformations as to keep the specific syntax of denials.

Example 3.1 Consider constraint theory Γ_1 from example 2.1 and update $U = \{m_1(a, b)\}$. We have

$$\begin{aligned} \text{After}^U(\Gamma_1) = \{ & \leftarrow m_1(X, Y) \wedge m_1(X, Z) \wedge Y \neq Z, \\ & \leftarrow m_1(X, Y) \wedge X = a \wedge Z = b \wedge Y \neq Z, \\ & \leftarrow X = a \wedge Y = b \wedge m_1(X, Z) \wedge Y \neq Z, \\ & \leftarrow X = a \wedge Y = b \wedge X = a \wedge Z = b \wedge Y \neq Z \}. \end{aligned}$$

Proposition 3.2 Let D be a database, U an update for D , and Γ a constraint theory. Then $D \models \text{After}^U(\Gamma)$ iff $D^U \models \Gamma$.

The result of **After** corresponds to what in Hoare's logic is called a weakest precondition.

Proposition 3.3 Let Γ be a consistently signed constraint theory and U an update. Then $\text{After}^U(\Gamma)$ is consistently signed.

Simplification involves an optimization phase which can employ (e.g.) the hypothesis that the state before the update is consistent. We indicate with $\text{Optimize}_\Theta(\Gamma)$ an operator which returns an optimized version of Γ employing the knowledge that a constraint theory Θ holds (also referred

to as “background knowledge” in the following). Any such operator must satisfy the following soundness condition.

For any D with $D \models \Theta$, $D \models \text{Optimize}_\Theta(\Gamma)$ iff $D \models \Gamma$

The **Optimize** operator should actually produce the formula which is the most efficient to evaluate, among those that satisfy the above condition. We may also assume that **Optimize** is idempotent and preserves the consistently signed property.

- $\text{Optimize}_\Theta(\text{Optimize}_\Theta(\Gamma)) = \text{Optimize}_\Theta(\Gamma)$
- Whenever $\Theta \cup \Gamma$ is consistently signed, so is $\text{Optimize}_\Theta(\Gamma)$.

An **Optimize** operator is described in [6] in terms of proof procedures that remove, from the input constraint theory, all denials and literals that can be proved redundant. In the end one obtains an output constraint theory with a minimized number of literals, which in most cases is a good approximation of the “most efficient” theory to evaluate. These details are not central to the present paper, and we will simply assume that there exists an optimization procedure, referred to as **Optimize**, having the above properties.

A condition to be checked in the current state to verify consistency of the updated state can therefore be produced by optimizing **After**'s output. In particular, one can check whether $\text{Optimize}_\Gamma(\text{After}^U(\Gamma))$ holds *before* executing an update U ; if it does not, the update is not even performed and expensive rollback operations to restore consistency can be avoided.

Example 3.2 For Γ_1 and U defined in examples 2.1 and 3.1, the optimized condition is calculated as $\{\leftarrow m_1(a, Y) \wedge Y \neq b\}$, that indicates that for a database D to keep consistency at source 1 after update U , husband a must not already be married to a wife different from b .

4 Integrity maintenance in DI systems

The situation for DI systems is different, as the sources are independent and the mediator is informed about an update after it is performed at the sources: simplified constraints are needed which apply in the updated state. Suppose as a first case that global consistency of a DI system $D = \langle F, \Delta, \Gamma \rangle$ was known to hold before an update U . An optimal test for consistency in the updated state can be produced as follows, with the background knowledge of all formulas known to hold in D^U .

$$\text{Optimize}_{\text{After}^{-U}(\Gamma) \cup \Delta \cup U \cup \text{After}^{-U}(\Delta)}(\Gamma) \quad (1)$$

Note that $\text{After}^{-U}(\Gamma)$ is a condition that evaluates in D^U in the same way in which Γ evaluates in D ; update U can be regarded as a set of formulas that will hold in D^U (easily rewritable as denials).

Example 4.1 Consider the DI system of example 2.1 and update $U = \{m_1(a, b)\}$. The task is now to find an optimal test which, under the given conditions, is equivalent to Γ_0 . Expression (1) above gives $\{\leftarrow m_2(a, Z) \wedge b \neq Z\}$. Clearly, this denial is much simpler to evaluate than the original Γ , and it is difficult imagine another denial satisfying the same semantics requirements which executes faster.

Adding the hypothesis that the domains of husbands in either source are disjoint ($\leftarrow m_1(X, Y) \wedge m_2(X, Y)$) would make formula (1) amount to true, i.e., under this assumption, U can never introduce global inconsistency.

As the sources are independent or autonomous, we need additional machinery to maintain global consistency. We now investigate the use of a correction table maintained at the global level. This table is expected to be small as it should only contain virtual repair actions to the sources. Global consistency can be obtained by maintaining a (possibly minimal) correction table. Alternative versions of the global predicates are then defined that take the corrections into account; this is straightforward and will not be considered in this paper.

Definition 4.1 Given a DI system $D = \langle F, \Delta, \Gamma \rangle$, a correction table for D is an update R to D so that $D^R \models \Gamma \cup \Delta$; R is minimal when there is no other $R' \subset R$ which is a correction table for D .

Trivially, for a consistent DI system D and an update U for which D^U is inconsistent, $\neg U$ is a correction table (although not necessarily a minimal one). Clearly, \emptyset is the only minimal correction table for a consistent DI system.

The relationship between correction tables and simplification is indicated by the following property. Notice that an existing correction table R' is assumed, and Θ is the collection of background knowledge available.

Proposition 4.2 Let $D = \langle F, \Delta, \Gamma \rangle$ be a DI system, U an update, and R' a correction table for D . Then R is a correction table for D^U iff

$$D^U \models \text{Optimize}_{\Theta}(\text{After}^R(\Gamma \cup \Delta))$$

where Θ is a set of formulas with $D^U \models \Theta$, namely $\Theta = \text{After}^{-U \circ R'}(\Gamma) \cup \Delta \cup U \cup \text{After}^{-U}(\Delta) \cup \text{After}^{-U \circ R'}(\Delta)$.

5 Maintenance of correction tables

We indicate a nondeterministic algorithm that can produce all possible, minimal correction tables and discuss below how it may be adapted to practical cases.

Definition 5.1 Let D be a DI system and Σ a constraint theory over source and built-in predicates. Let

$\text{Ground}(\Sigma)$ be the set of all ground instances of denials in Σ over constants in D ; $\text{Falsifiers}(D, \Sigma)$ is defined as $\{\sigma \in \text{Ground}(\Sigma) \mid D \not\models \sigma\}$. We define $\text{PotentialCorrections}(D, \Sigma)$ as $\{\text{neg}(L) \mid (\leftarrow \dots \wedge L \wedge \dots) \in \text{Falsifiers}(D, \Sigma)\}$. A potential correction set for Σ in D is a subset $C \subseteq \text{PotentialCorrections}(D, \Sigma)$ such that each denial of $\text{Falsifiers}(D, \Sigma)$ contains some literal M with $\text{neg}(M) \in C$; the set of all potential correction sets for Σ in D is indicated $\text{PCSets}(D, \Sigma)$.

The sets defined above are finite since every component is finite and Σ is range restricted; they can be found by posing Σ as a query to D followed by straightforward processing of the result.

Picking an element S of $\text{PCSets}(D, \Sigma)$ does not necessarily give rise to a correction table for Σ , as other falsifiers may show up in the “corrected database” D^S . The following special case is an exception.

Proposition 5.2 Let $D = \langle F, \Delta, \Gamma \rangle$ be a consistently signed DI system. Then $\text{PCSets}(D, \Gamma)$ consists of correction tables for D , and any minimal correction table for D is member of $\text{PCSets}(D, \Gamma)$.

The general case is handled by the following nondeterministic algorithm, that also considers an update and a correction table for the database before this update.

Algorithm 1

Input: DI system $D = \langle F, \Delta, \Gamma \rangle$, update U , correction table R' for D .

Output: A correction table for D^U .

1. let $R := \emptyset$
2. let $\Theta := \text{After}^{-U \circ R'}(\Gamma) \cup U \cup \Delta \cup \text{After}^{-U}(\Delta) \cup \text{After}^{-U \circ R'}(\Delta)$
3. let $\Sigma := \text{Optimize}_{\Theta}(\text{After}^R(\Gamma \cup \Delta))$
and $C := \text{PotentialCorrections}(D^U, \Sigma)$
4. if $C = \emptyset$, goto 7
5. let $R := R \cup \{L\}$ where $L \in C \setminus R$, $\text{neg}(L) \notin R$, and $D^U \not\models L$
6. if there is no such L , abort, otherwise goto 3
7. let $R := R \setminus \{L\}$ for some $L \in R$ such that
 $D^U \models \text{Optimize}_{\Theta}(\text{After}^R(\Gamma \cup \Delta))$
8. if there is no such L , return R
9. goto 7

The algorithm uses variable R to build a correction table adding elements one by one from C , that represent the “bugs” that remain to be fixed. Notice that this C is calculated for each iteration to take into account the new bugs derived from previous bug fixes. Nondeterminism in the selection of L in step 5 ensures that all possibilities are tried out; the possible abort in step 6 rules out wrong choices. However, there will be at least one solution, namely $R = \neg U \circ R'$. Steps 7 to 9 nondeterministically remove elements as long as possible without destroying the property that R is a correction table; in this way the algorithm outputs only (and all) minimal correction tables.

Example 5.1 In our running example, assume a DI system D with (minimal) correction table $R' = \emptyset$ and consider update $U_1 = \{m_1(a, b)\}$. In case $m_2(a, c)$ is true in the second source, the algorithm produces two different

new minimal correction tables for D^U : $\{\neg m_1(a, b)\}$ and $\{\neg m_2(a, c)\}$.

Assuming instead another DI system D' with minimal correction table $R' = \{m_2(a, b)\}$ and the same update U , R' is output unchanged as the sole minimal table for the updated DI system D'^U . To see this, notice that $S_2^{R'}$ satisfies its local ICs, where S_2 refers to the second source, so S_2 cannot contain any other tuple of the form $m_2(a, -)$.

6 Discussion

In practice it is infeasible to maintain the collection of all (minimal) correction tables as their number may be exponential in the size of the original database [4]. The use of correction tables seems best suited for cases where there is *one* natural way of correcting a DI system. This can be the case in DI systems with the opportunity to check proposed correction literals interactively with an expert user. In this case, step 5 should be extended by querying the user whether the chosen L is relevant and the system should avoid to suggest correcting information already verified; in this system, minimization in steps 7–9 should be removed. The system of [7] appears to be a special case of this pattern. It may also be envisaged that problematic updates be collected over time and verified with a user only when their number goes beyond a certain threshold.

Another option is to give preference to newer information (if this is available across sources), as suggested by several other authors, or to exploit user-defined preferences for automatic conflict resolution.

Although the number of needed corrections as well as the number of iterations in algorithm 1 is expected to be relatively small, it may appear to have a high computational cost; still, it can be optimized in several ways. For example, updates that, by some simplification approach, are known not to violate the integrity constraints will not change the correction table. Conversely, by looking at the sign of a fact in the update, it is easy to see which constraints of Γ and Δ need to be considered.

Furthermore, the use of parameters [6] may make it possible to unfold (or partially evaluate) the algorithm into a decision tree where choices are made from queries posed to the source. For instance, in example 5.1, a , b and c can be considered as parameters and their actual values can be plugged in at runtime.

References

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

- [2] Leopoldo E. Bertossi and Jan Chomicki. Query answering in inconsistent databases. In Jan Chomicki, Ron van der Meyden, and Gunter Saake, editors, *Logics for Emerging Applications of Databases*, pages 43–83. Springer, 2003.
- [3] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.
- [4] Jan Chomicki and Jerzy Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In Leopoldo E. Bertossi, Anthony Hunter, and Torsten Schaub, editors, *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*, pages 119–150. Springer, 2005.
- [5] Henning Christiansen and Davide Martinenghi. Simplification of integrity constraints for data integration. In Dietmar Seipel and Jose Maria Turull Torres, editors, *Foundations of Information and Knowledge Systems, Third International Symposium (FoIKS 2004)*, volume 2942 of *Lecture Notes in Computer Science*, pages 31–48. Springer, 2004.
- [6] Henning Christiansen and Davide Martinenghi. On simplification of database integrity constraints. *Fundamenta Informaticae*, 71:1–47, 2006.
- [7] Suzanne M. Embury, Sue M. Brandt, John S. Robinson, Iain Sutherland, Frank A. Bisby, W. A. Gray, Andrew C. Jones, and Richard J. White. Adapting integrity enforcement techniques for data reconciliation. *Inf. Syst.*, 26(8):657–689, 2001.
- [8] Davide Martinenghi, Henning Christiansen, and Hendrik Decker. Integrity checking and maintenance in relational and deductive databases – and beyond. In *Intelligent Databases: Technologies and Applications*. Idea Group Inc., 2006. To appear.
- [9] Jean-Marie Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18:227–253, 1982.
- [10] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I & II*. Computer Science Press, 1988/89.
- [11] Jeffrey D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.
- [12] Jef Wijsen. On condensing database repairs obtained by tuple deletions. In *DEXA Workshops*, pages 849–853. IEEE Computer Society, 2005.