Effectively Updatable Conjunctive Views

Effectively Updatable Conjunctive Views

(Extended Abstract)

Enrico Franconi and Paolo Guagliardo

KRDB Research Centre, Free University of Bozen-Bolzano, Italy
{franconi,guagliardo}@inf.unibz.it

1 Introduction

The *view update problem* [1] consists in finding suitable ways of consistently and univocally propagating the changes introduced into a set of the view relations to the underlying database relations over which the view relations are defined. This can be formalised within a general framework [1,7] where a *view* is a function that associates instances of a database schema with instances of a view schema, which has a constructive characterisation when each view symbol is defined in terms of the database symbols in a concrete query language. A view is *updatable* if the changes introduced into the view relations by updates can be unambiguously propagated back to the underlying database relations over which the view relations are defined. This is possible whenever the view is *invertible*, but invertibility in itself is not enough for practical purposes, as it merely indicates that the database instances functionally depend on the corresponding view instances. What is actually needed is in fact a constructive characterisation of the inverse, obtained by finding an *exact rewriting* of each database symbol in terms of the view symbols, expressed in a query language that is not necessarily the same as the one used for defining the view symbols.

In the context of relational databases, the study of the invertibility of views has focused on very restricted settings. Cosmadakis and Papadimitriou [2] limit their investigation to only two view relations defined by projections over a single database relation, which was recently generalised in [7] to an arbitrary number of view relations defined by acyclic projections, but still over a database consisting of a single relation. Moreover, the set of integrity constraints considered on the database schema has been limited to functional and join dependencies in [2] and full embedded dependencies in [7].

In this paper, we consider a setting where the view symbols are defined by conjunctive queries (CQs) and we show that, when the integrity constraints on the database schema are stratified embedded dependencies [3], a view is invertible precisely if each database symbol has an exact rewriting given by a CQ over the view schema. We then discuss how such rewritings can be effectively found using the Chase & Backchase algorithm [4]. As a special case, in combination with the general criterion for the translatability of view updates we introduced in [7], our results settle the long-standing open issue pointed out in [2] of how to solve the view update problem for a multi-relational database with view relations that are projections of joins of the database relations.

Outline. The rest of the paper is organised as follows: after some preliminaries in Sec. 2, in Sec. 3 we recall and summarise the framework introduced in [7]; in Sec. 4 we show how view updatability can be checked, and rewritings effectively found, when the view symbols are defined by CQs in a multi-relational database under stratified embedded dependencies; we conclude in Sec. 5 by pointing out future research directions.

2 Preliminaries

A *schema* is a finite set of relation symbols. Let **dom** be an arbitrary (possibly infinite) set of domain values. An *instance* I of a schema $\mathcal{S}$ maps each relation symbol S in $\mathcal{S}$ to a relation S^I on **dom** of appropriate arity, called the *extension* of S under I. The set of elements of **dom** that occur in an instance I is called the *active domain* of I and is denoted by $\mathbf{adom}(I)$. An instance is finite when its active domain is, and we always assume instances to be finite. We consider a *database schema* $\mathcal{R}$ of *database symbols* and a *view schema* $\mathcal{V}$ of *view symbols* not occurring in $\mathcal{R}$. A *database state* is an instance $I_\mathcal{R}$ of $\mathcal{R}$ and a *view state* is an instance $I_\mathcal{V}$ of $\mathcal{V}$. The set of all database states (resp., view states) is denoted by $\mathbf{R}$ (resp., $\mathbf{V}$). The *disjoint union* $I_\mathcal{R} \uplus I_\mathcal{V}$ of a database state $I_\mathcal{R}$ and a view state $I_\mathcal{V}$ is the instance, called a *global state*, of the *global schema* $\mathcal{R} \cup \mathcal{V}$, with active domain $\mathbf{adom}(I_\mathcal{R}) \cup \mathbf{adom}(I_\mathcal{V})$ and associating each relation symbol S in $\mathcal{R} \cup \mathcal{V}$ with $S^{I_\mathcal{R}}$ if $S \in \mathcal{R}$ and with $S^{I_\mathcal{V}}$ otherwise.

We consider a satisfiable (finite) set Σ of *global constraints* over $\mathcal{R} \cup \mathcal{V}$, consisting of a set $\Sigma_\mathcal{R}$ of *database constraints* over $\mathcal{R}$ and a set $\Sigma_{\mathcal{RV}}$ of *interschema constraints* over $\mathcal{R} \cup \mathcal{V}$. The set $\Sigma_{\mathcal{RV}}$ consists of exactly one formula of the form $\forall \overline{x} \left(V(\overline{x}) \leftrightarrow \phi(\overline{x}) \right)$ for each $V \in \mathcal{V}$, where $\phi(\overline{x})$ mentions only database symbols and is called a *definition* of V in terms of $\mathcal{R}$. Note that, as $\mathcal{R}$ and $\mathcal{V}$ are disjoint, every instance of $\mathcal{R} \cup \mathcal{V}$ satisfying Σ has the form $I_\mathcal{R} \uplus I_\mathcal{V}$ where $I_\mathcal{R}$ and $I_\mathcal{V}$ are a database state and a view state, respectively. A view state $I_\mathcal{V}$ (resp., database state $I_\mathcal{R}$) is *Σ-consistent* (or *globally consistent* or *consistent with the global constraints*) if there is a database state $I_\mathcal{R}$ (resp., view state $I_\mathcal{V}$) such that $I_\mathcal{R} \uplus I_\mathcal{V}$ satisfies the global constraints Σ. Observe that every legal database state (i.e., one that satisfies the database constraints $\Sigma_\mathcal{R}$) is globally consistent. We denote the set of Σ-consistent view states (resp., database states) by $\mathbf{V}_\Sigma$ (resp., $\mathbf{R}_\Sigma$).

We say that $\mathcal{V}$ *determines* $\mathcal{R}$ *under* Σ (written $\mathcal{V} \twoheadrightarrow_\Sigma \mathcal{R}$) if, for every $I_\mathcal{V}$ and $I_\mathcal{R}, I'_\mathcal{R}$, it is the case that $I_\mathcal{R} = I'_\mathcal{R}$ whenever $I_\mathcal{R} \uplus I_\mathcal{V} \models \Sigma$ and $I'_\mathcal{R} \uplus I_\mathcal{V} \models \Sigma$. In other words, models of Σ that agree on the extension of the view symbols also agree on the extension of the database symbols, which means that in every model of Σ the latter functionally depends on former. Clearly, under the assumptions we made on the global constraints Σ, it is always the case that $\mathcal{R} \twoheadrightarrow_\Sigma \mathcal{V}$, and we refer to the corresponding functional mapping $f \colon \mathbf{R}_\Sigma \to \mathbf{V}_\Sigma$, associating each Σ-consistent database state with a Σ-consistent view state, as the *view from $\mathcal{R}$ to $\mathcal{V}$ induced by Σ*. In the rest of the paper, unless specified otherwise, whenever we say “a view” we refer to the (one and only) view from $\mathcal{R}$ to $\mathcal{V}$ induced by a set of global constraints Σ as above.

3 The View Update Framework

In this section we briefly recapitulate the general view update framework previously introduced in [7]. A *view update* is a function $u\colon \mathbf{V} \to \mathbf{V}$ associating each view state with another, possibly the same. Given a view update that modifies the current view state, we want to modify the database state accordingly so as to reflect *exactly* the changes introduced into the view state. For this to be possible in an unambiguous way, the view must be *updatable*, and the view update *translatable*, as we formally define next.

Definition 1 (Updatability). *A view is* updatable *if $\mathcal{V} \twoheadrightarrow_{\Sigma} \mathcal{R}$.*

Note that a view in our sense is always surjective, and $\mathcal{V} \twoheadrightarrow_{\Sigma} \mathcal{R}$ further implies injectivity [7], hence the notion of updatability coincides with invertibility.

Definition 2 (Translatability). *Let u be a view update and let $I_{\mathcal{V}} \in \mathbf{V}_{\Sigma}$. We say that u is* translatable on $I_{\mathcal{V}}$ *if $u(I_{\mathcal{V}}) \in \mathbf{V}_{\Sigma}$.*

A translatable view update leads to view states in the image of the view f, which are therefore reachable from some database state by means of f. In such a case, when the view is updatable, the changes introduced into the view state $I_{\mathcal{V}}$ by the view update u, resulting in the updated view state $I'_{\mathcal{V}} = u(I_{\mathcal{V}})$, can be univocally pushed back by updating the database to the new state $f^{-1}(I'_{\mathcal{V}})$.

However, the fact that $\mathcal{V}$ determines $\mathcal{R}$ under Σ, even though it ensures that a view is invertible, does not actually provide a constructive characterisation of its inverse. In other words, $\mathcal{V} \twoheadrightarrow_{\Sigma} \mathcal{R}$ guarantees that the extension of the database symbols functionally depends on that of the view symbols, but it says nothing on *how* the former is to be obtained from the latter. In order to effectively compute the inverse of a view, we must be able to explicitly express each database symbol $R \in \mathcal{R}$ in terms of the view symbols $\mathcal{V}$ by means of a formula ψ, called an *exact rewriting of R in terms of $\mathcal{V}$ under Σ*, mentioning only view symbols and such that $\Sigma \models \forall \overline{x} \left(R(\overline{x}) \leftrightarrow \psi(\overline{x}) \right)$.

Definition 3. *A view is* effectively updatable *if each database symbol has an exact rewriting in terms of the view symbols under Σ.*

Whenever a view update results in a view state $I_{\mathcal{V}}$ in $\mathbf{V}_{\Sigma}$, if a view is effectively updatable the changes can be propagated to the database state $f^{-1}(I_{\mathcal{V}})$ by computing the extension of each database symbol from its rewriting in terms of the view symbols. But at this point, the question is: How do we ascertain whether a view state belongs in fact to $\mathbf{V}_{\Sigma}$? The solution consists in constructing what we call the *$\mathcal{V}$-embedding* $\widetilde{\Sigma}_{\mathcal{V}}$ of Σ, which is obtained from Σ by replacing every occurrence of each $R \in \mathcal{R}$ with its rewriting in terms of $\mathcal{V}$. The resulting set of constraints mentions only view symbols and, as it turns out, is satisfied exactly by all and only the view states in $\mathbf{V}_{\Sigma}$. Thus, checking for the translatability of a view update amounts to checking whether the updated view state satisfies $\widetilde{\Sigma}_{\mathcal{V}}$.

Theorem 1 ([7]). *Let f be an effectively updatable view, let u be a view update, and let $I_{\mathcal{V}} \in \mathbf{V}_{\Sigma}$. Then, u is translatable on $I_{\mathcal{V}}$ if and only if $u(I_{\mathcal{V}}) \models \widetilde{\Sigma}_{\mathcal{V}}$.*

Whether a view update u is translatable on a view state $I_{\mathcal{V}}$ satisfying $\widetilde{\Sigma}_{\mathcal{V}}$ can be checked in polynomial time in the size of $u(I_{\mathcal{V}})$, which is the data complexity of testing whether a finite relational structure is a model of a FOL theory.

Summing up, for the above machinery to work it is essential to find a rewriting of each database symbol in terms of the view symbols, in order to build the $\mathcal{V}$-embedding of Σ (and thus checking for the translatability of updates) and to propagate the changes introduced by translatable view updates (by computing the extension of the database symbols from that of the view symbols). Note that checking whether a view is invertible, and eventually computing its inverse, is an operation performed once for all offline.

4 Conjunctive Views

In this section, we settle the long-standing open issue pointed out in [2], namely how to solve the view update problem in a multi-relational database with views that are projections of joins of relations, and we do so in a more general setting where the view symbols are defined by CQs and the constraints on the database schema are embedded dependencies satisfying appropriate restrictions.

Example 1. Let $\mathcal{R} = \{R_1, R_2\}$ and $\Sigma_{\mathcal{R}}$ consist of the following embedded dependencies (in this case, inclusion and functional dependencies):[1]

$$R_1(x,y,z) \rightarrow \exists v,w\ R_2(z,v,w)\ , \tag{1a}$$
$$R_2(z,v,w) \rightarrow \exists x,y\ R_1(x,y,z)\ , \tag{1b}$$
$$R_1(x,y,z) \wedge R_1(x',y,z') \rightarrow z = z'\ , \tag{1c}$$
$$R_2(z,v,w) \wedge R_2(z,v',w') \rightarrow v = v'\ . \tag{1d}$$

Let $\mathcal{V} = \{V_1, V_2, V_3\}$ and $\Sigma_{\mathcal{RV}}$ consist of the following view definitions:

$$V_1(x,y) \leftrightarrow \exists z\quad R_1(x,y,z)\ , \tag{2a}$$
$$V_2(y,z,v) \leftrightarrow \exists x,w\ R_1(x,y,z) \wedge R_2(z,v,w)\ , \tag{2b}$$
$$V_3(z,w) \leftrightarrow \exists v\quad R_2(z,v,w)\ . \tag{2c}$$

The embedded dependencies we consider are required to satisfy the condition known as *stratification* [3], which is based on the notion of *chase graph*: The chase graph of a set of embedded dependencies Σ has the dependencies in Σ as nodes and, for $\alpha, \beta \in \Sigma$, has an edge from α to β if and only if, intuitively, firing α may cause β to fire as well (refer to [3] for the formal definition). Then, Σ is *stratified* if the set of dependencies in every cycle of its chase graph is *weakly acyclic* [6,5]. Note that every weakly acyclic set of dependencies is also stratified. Indeed, $\Sigma_{\mathcal{R}}$ of Example 1 is weakly acyclic and, in turn, stratified.

Our main result establishes that, when the embedded dependencies over the database schema are stratified, the view is invertible precisely if each database symbol has an exact rewriting as a conjunctive query over the view schema.

[1] Universal quantifiers are omitted.

Algorithm 1

```
INPUT:  an atomic query over $\mathcal{R}$, a view schema $\mathcal{V}$, a set of embedded dependencies
        $\Sigma$ over $\mathcal{R} \cup \mathcal{V}$.
OUTPUT: an exact CQ rewriting of $q$ in terms of $\mathcal{V}$ under $\Sigma$, if any, or $\perp$ otherwise.
1: function Rewrite($q, \mathcal{V}, \Sigma$)
2:     for each subquery $q'$ of chase($q, \Sigma$) over $\mathcal{V}$ do
3:         repeat
4:             if $\exists$ containment mapping from $q$ to $q'$ return $q'$
5:             set $q'$ to chase-step($q', \Sigma$)
6:         until no further chase step applies
7:     end for
8:     return $\perp$
9: end function
```

Theorem 2. *Let $\Sigma = \Sigma_{\mathcal{R}} \cup \Sigma_{\mathcal{RV}}$, where $\Sigma_{\mathcal{R}}$ consists of stratified embedded dependencies and each $V \in \mathcal{V}$ is defined in $\Sigma_{\mathcal{RV}}$ by a CQ. Then, $\mathcal{V} \twoheadrightarrow_{\Sigma} \mathcal{R}$ if and only if each $R \in \mathcal{R}$ has an exact CQ rewriting in terms of $\mathcal{V}$ under Σ.*

The *Chase and Backchase* (C&B) [4] is an algorithm that enumerates the exact CQ-rewritings of a CQ under constraints. More precisely, given two schemas $\mathcal{S}$ and $\mathcal{T}$, a set of embedded dependencies Γ over $\mathcal{S} \cup \mathcal{T}$, and an input CQ q over $\mathcal{S}$, the C&B outputs all the CQs over $\mathcal{T}$ which are equivalent to q under Γ. The C&B is sound and complete, in the sense that it returns all and only the CQs into which the input CQ can be rewritten (up to homomorphic equivalence) under the given constraints, whenever the chase is guaranteed to terminate, which is the case, e.g., for stratified sets of dependencies. Obviously, the fact that the output of the C&B is empty for q does not mean that q has no rewriting in terms of $\mathcal{T}$ under Γ, but simply that its rewriting, if any, is not a CQ.

We can use the C&B to look for the rewritings we are interested in. For each $R \in \mathcal{R}$, consider the atomic query $q(\overline{x}) = R(\overline{x})$ and proceed as follows:

Chase. Chase q with Σ until no further chase step applies. The resulting query is the so-called *universal plan U*.

Backchase. Every subquery of U over $\mathcal{V}$ (i.e., a set of $\mathcal{V}$-atoms from U mentioning all of q's free variables) is a candidate rewriting of q. Chase each candidate q' with Σ step-by-step until no further chase step applies, and at each new step in the chase sequence check whether a containment mapping from the original query q can be found. If that is the case, then q' is a rewriting of q.

The above is described in more detail in Algorithm 1 and illustrated in our running example.

Example 2. Chasing the query $q(x, y, z) = R_1(x, y, z)$ with Σ of Example 1 gives the following universal plan:

$$U(x, y, z) = \exists v, w\ R_1(x, y, z) \wedge R_2(z, v, w) \wedge V_1(x, y) \wedge V_2(y, z, v) \wedge V_3(z, w)\ .$$

A candidate rewriting of $R(x, y, z)$ in terms of $\mathcal{V}$ is the subquery $q'(x, y, z) = \exists v\ V_1(x, y) \wedge V_2(y, z, v)$, that chased with the left-to-right TGDs from (2a) and (2b) yields the following:

$$
\begin{aligned}
q''(x, y, z) = {} & \exists v, z', x', w \\
& V_1(x, y) \wedge V_2\,(y\ , z, v) \wedge R_1(x, y, z') \wedge R_1(x', y, z) \wedge R_2(z, v, w)\ .
\end{aligned}
$$

A further chase step with (1c) gives $z' = z$, and therefore we can find a containment mapping (the identity) from the original query q to q''. Thus, the rewriting of $R_1(x, y, z)$ is $\exists v\ V_1(x, y) \wedge V_2(y, z, v)$. Similarly, we also have that $R_2(z, v, w)$ can be rewritten in terms of $\mathcal{V}$ as $\exists y\ V_2(y, z, v) \wedge V_3(z, w)$.

As it turns out, the constraints Σ considered in Theorem 2 are stratified, and this ensures that if an exact CQ-rewriting of $R(\overline{x})$ cannot be found by means of Algorithm 1, then R cannot be expressed at all in terms of the view symbols.

Theorem 3. *Let Σ be as in Theorem 2. Then, the procedure* Rewrite *of Algorithm 1 is sound and complete for finding the exact rewriting of each database symbol in terms of the view symbols under Σ. Moreover, $\mathcal{V} \twoheadrightarrow_{\Sigma} \mathcal{R}$ if and only if* Rewrite$(q, \Sigma, \mathcal{V}) \neq \bot$ *for every atomic query q over $\mathcal{R}$.*

The results presented above extend the setting of [7], consisting of *only one* database symbol, view symbols defined by acyclic projections and database constraints given by full dependencies, which is a special case where the rewriting is known to be the join, rather than a generic CQ.

5 Outlook

We conclude by pointing out and briefly discussing possible research directions that would be interesting to pursue and investigate further.

- Consider views defined by queries expressed in languages beyond CQs. A first natural candidate is the class of unions of conjunctive queries.
- Consider different constraints on the database schema. Several sufficient conditions for chase termination have been proposed, e.g., *super-weak acyclicity* [8], *safety* and *inductive restriction* [9], some of which extend stratification, while some other are incomparable with it. The question is whether the global constraints satisfy such conditions, as is the case for stratification when views are defined by CQs and database constraints are stratified embedded dependencies.
- Consider constraints also on the view schema. We can allow a set of view constraints $\Sigma_{\mathcal{V}}$ for which $\Sigma_{\mathcal{R}} \cup \Sigma'_{\mathcal{V}}$ is a set of stratified embedded dependencies, where $\Sigma'_{\mathcal{V}}$ is the set of constraints over $\mathcal{R}$ obtained from $\Sigma_{\mathcal{V}}$ by replacing every occurrence of each *view symbol* with its CQ-definition (given in $\Sigma_{\mathcal{R}\mathcal{V}}$) in terms of the database symbols. What is interesting to understand is the shape that such view constraints must have in order to satisfy the above condition.

References

1. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM Transactions on Database Systems 6(4), 557–575 (Dec 1981)
2. Cosmadakis, S.S., Papadimitriou, C.H.: Updates of relational views. Journal of the Association for Computing Machinery 31(4), 742–760 (Oct 1984)
3. Deutsch, A., Nash, A., Remmel, J.: The Chase revisited. In: Proc. of PODS 2008. pp. 149–158. ACM (2008)
4. Deutsch, A., Popa, L., Tannen, V.: Query reformulation with constraints. SIGMOD Record 35(1), 65–73 (Mar 2006)
5. Deutsch, A., Tannen, V.: Reformulation of XML queries and constraints. In: Proc. of ICDT 2003. LNCS, vol. 2572, pp. 225–241. Springer (2002)
6. Fagin, R., Kolaitis, P., Miller, R., Popa, L.: Data exchange: Semantics and query answering. In: Proc. of ICDT 2003. LNCS, vol. 2572, pp. 207–224. Springer (2002)
7. Franconi, E., Guagliardo, P.: On the translatability of view updates. In: Proc. of AMW. CEUR Workshop Proceedings, vol. 866, pp. 154–167. CEUR-WS.org (2012)
8. Marnette, B.: Generalized schema-mappings: from termination to tractability. In: Proc. of PODS 2009. pp. 13–22. ACM, New York, NY, USA (2009)
9. Meier, M., Schmidt, M., Lausen, G.: On chase termination beyond stratification. Proceedings of the VLDB Endowment 2(1), 970–981 (Aug 2009)