Reformulation of XML Queries and Constraints

Alin Deutsch[1] and Val Tannen[2]

[1] UC San Diego, deutsch@cs.ucsd.edu
[2] University of Pennsylvania, val@cis.upenn.edu

Abstract. We state and solve the query reformulation problem for XML publishing in a general setting that allows mixed (XML and relational) storage for the proprietary data and exploits redundancies (materialized views, indexes and caches) to enhance performance. The correspondence between published and proprietary schemas is specified by views in both directions, and the same algorithm performs rewriting-with-views, composition-with-views, or the combined effect of both, unifying the Global-As-View and Local-As-View approaches to data integration. We prove a completeness theorem which guarantees that under certain conditions, our algorithm will find a minimal reformulation if one exists. Moreover, we identify conditions when this algorithm achieves optimal complexity bounds. We solve the reformulation problem for constraints by exploiting a reduction to the problem of query reformulation.

1 Introduction

The problem of **query reformulation** is a very general one: given two schemas P and S and a correspondence CR between them, and given a query Q formulated in terms of P, find a query X formulated in terms of S that is equivalent to Q modulo the correspondence CR. Reformulation algorithms have many uses in database technology, for example in data integration where P is the *global* integrated schema and S gathers the *local* schemas of the actual data sources, or in schema evolution where P is the old schema and S is the new schema.

In this paper our motivation and specific challenges come from **XML publishing**, where P is the *public* XML schema and S is the *storage* schema of the proprietary data from which selected portions are published. Typically, the proprietary data resides in relational databases (RDB) and lately also in native XML document storage (e.g., if acquired through XML exchange). Clients formulate queries against the public XML schema (in our case in XQuery [33]) and the publishing system must *reformulate* these into queries on the storage schema data in order to answer them.

A central problem is how to model the schema correspondence CR. Data integration systems use one of two approaches for the analogous problem [20, 22]: "Global-As-View" (GAV) and "Local-As-View" (LAV) with the views themselves (sometimes called *mappings*) expressed in a query language. We shall use these acronyms but keep in mind that for us

GAV views : storage ⟶ public LAV views : public ⟶ storage

In fact, neither of these two approaches used in *isolation* is flexible enough for our problem. The GAV approach is convenient for **hiding portions of the proprietary data**: the view definition can simply project/select them away. This cannot be done in a LAV approach, since the view's input is in this case the published data, from which the hidden information is missing. On the other hand, we will also want to tune the performance of the publishing system by, e.g., caching query results or redundantly storing some of the native XML data in relational databases in order to exploit the more mature relational technology. The resulting **redundancies in the stored data** can be easily exploited in the LAV approach and will typically lead to multiple reformulations. [3] However, existing techniques for the GAV approach do not handle such redundancies properly (see related work). We conclude that in common XML publishing scenarios we need schema correspondences specified using a *combination* of both kind of views (GLAV), each of them a mapping from a portion of the storage schema to a portion of the public schema, or conversely. To facilitate design and administration tasks, agreeing with [8] that XML encodings of relational schemas are easily understood and used, we shall assume that these views are expressed in XQuery.

Finally, we consider integrity constraints on both the public and the storage schema. (Note that the presence of constraints will never reduce but may often *expand* the space of possible reformulations.) While much is known about relational constraints, XML constraint formalisms are still "under construction". We follow here our proposal [12] for a class of **X**ML **I**ntegrity **C**onstraints (`XIC`) whose expressive power captures a considerable part of XML Schema [32, 4] including keys and "keyrefs" and quite a bit beyond.

Therefore, in this paper we study the following problem:

Given:

– the public schema P as XML, with constraints
– the storage schema S: mixed, RDB + XML, with constraints
– the client query Q formulated over P in XQuery
– the schema correspondence CR between P and S formulated as
    - a (simple) encoding of RDB into XML
    - mappings(views) between portions of P and S in both directions (GLAV)

Find:

– one or more queries X formulated over S, such that
– X is equivalent to Q under CR

Our **approach** to query reformulation is to "compile" the XML reformulation problem into a relational reformulation problem and then use an algorithm that we have proposed earlier together with Lucian Popa [10]. The different ingredients of this strategy are sketched in the following steps 1–5.

Step 1 We encode the stored relational schemas into XML (pick one of several straightforward encodings). Then, the DB administrator can define map-

[3] When our reformulation algorithm produces multiple candidates, these should be further compared using application-specific cost models. This important step is outside the scope of this paper (but see [27]).

pings RDB→XML or XML→RDB just by writing them in XQuery. Thus, the schema correspondence is given by several **XQuery views** (in both directions). We also take integrity constraints into consideration, on the relational part as *disjunctive embedded dependencies* (DEDs), see [1, 14] and section 3, and on the XML part as `XIC`s, see section 4.2 and [12]. [4]

Step 2 Like [8, 25] we follow [16] in splitting **XQuery = navigation part + tagging template** corresponding to the two phases in the operational semantics of XQuery [33], see section 4.2. Previous research has addressed the efficient implementation of the second phase [30, 15]. Only the first phase depends on the schema correspondence so we focus on **reformulating the navigation part of XQueries**.

Step 3 We define a **generic relational encoding for XML** [5] whose schema we call $\mathcal{X}$ (see section 4.1). Then, the XML encoding (see Step 1) of the stored relational schema is captured by a set of DEDs relating these schemas to schema $\mathcal{X}$, as explained in section 4.3.

Step 4 We define a syntactic restriction of XQuery, the *behaved* queries, that are still very powerful (see section 4). We give algorithms that **translate**: (1) the navigation part (see Step 2) of a behaved XQuery into a relational union of conjunctive queries over $\mathcal{X}$, call it B, (2) the behaved XQuery views in the schema correspondence (see Step 1) into sets of relational DEDs over $\mathcal{X}$ (see Step 3), and (3) the `XIC`s from both schemas (see Step 1) also into sets of relational DEDs over $\mathcal{X}$.

Step 5 We now have a relational query B (see Step 5) that needs to be reformulated modulo equivalence under the set of all relational constraints coming from Steps 1, 3, and 4. For this we use the **C&B algorithm** [10]. We prove new theorems that show that our algorithm is indeed **complete** in that it finds all "minimal" reformulations (see sections 3 and 4.4).

Why "minimal"? Note that in general a query has infinitely many reformulations just by trivially adding repeating scans (eg., items in the from clause). We call a reformulation **minimal** (see section 3) if it performs a minimal number of scans over source data, in the sense that we cannot remove a scan without compromising equivalence to the original query. Note also that if a query has any reformulation then it will have a minimal one as well.

Our approach is summarized in Figure 1 which happens to also be describing the architecture of the MARS (mixed and redundant storage) system that implements it (more in section 6).

The **constraint reformulation** problem also arises naturally in the XML publishing scenario. If a certain constraint d on the published data is desired, an administrator may be able to achieve this by enforcing *additional* constraints on the storage data. But which ones? We could guess, and test as follows: compile

[4] Because of the encoding of RDB in XML we can also use `XIC`s for constraints between the RDB and XML parts.

[5] Interestingly, in a mixed RDB + XML situation we encode RDB in XML to make view and query specification user-friendly, but then we encode XML in RDB for the automated query processing!

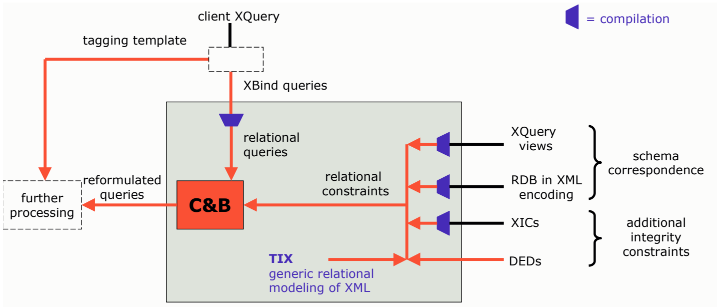

Fig. 1. MARS architecture

the schema correspondence and the storage constraints into a set Δ of relational constraints; compile the desired XML constraints on the published schema into a set D of relational constraints; then ask if $\Delta \models D$, using the chase to test it. But there may be a better way. Namely, *reformulate* d into a storage constraint δ that is equivalent to d modulo the schema correspondence. It may then be easier to redesign the storage constraints in order to enforce δ (hence d). In section 5 we describe one approach to such reformulation.

2 Contributions and related work

The conceptual contribution of this work to the XML publishing research topic is a uniform solution to the problem of finding **minimal reformulations of XQueries**, when the schema correspondence is given by a combination of GAV- and LAV-style XQuery views. Our solution allows mixed storage (RDB and XML), and integrity constraints on both the public and storage schemas. Our approach unifies the LAV and GAV data integration scenarios by achieving the combined effect of rewriting-with-views, composition-with-views and minimization. Moreover, we show how to **apply our algorithm to constraint reformulation** by exploiting the inter-reducibility of the problems of query containment and dependency implication (section 5). All of this is made possible by the following technical contributions.

We **reduce** this XML problem to a similar problem involving only relational queries and relational dependencies. We give translation algorithms for this reduction (see step 5 in section 1). We prove a **relative completeness theorem** for the translation (Theorem 2) that says in essence that any existing solution of the XML problem can be recovered from some minimal relational reformulation that is a solution of the relational translated problem.

The translated problem consist of finding minimal reformulations of unions of conjunctive queries under sets of disjunctive embedded dependencies (section 3). We solve this problem with the C&B algorithm. This algorithm was introduced in [10] and extended in [14] to also deal with unions and disjunctions. A limited completeness theorem was shown in [10], for the case when the constraints correspond to just LAV views, no views in the reverse direction and no additional constraints on the schemas. In this paper we prove a much more general **C&B completeness theorem**, namely for any set of constraints that yield a terminating chase (Theorem 1). By combining Theorem 2 with Theorem 1, we conclude that our solution to the XML reformulation problem is **overall complete** (Corollary 1). Our completeness results hold only for the *behaved* queries (defined in section 4), and for *bounded* XML constraints (in section 4.4). In fact, the method is applicable to larger classes of queries, views and constraints, as long as we can compile them and apply the chase, being understood that we don't have completeness guarantees anymore. From a practical perspective, we argue that the features that we cover are in our experience the most common ones anyway.

The limitations of the method are not arbitrary. To **calibrate our results** we first show that checking minimality under dependencies is as hard as deciding query containment (Proposition 2). This allows us to use lower bounds from [12] on the containment of the navigation part of XQueries to show that the restrictions we have imposed are quite essential. Indeed, we conclude that even modest extensions of the class of behaved XQueries will make our algorithm **incomplete** (unless $NP = \Pi_2^p$). We also conclude that even modest use of *unbounded* XML constraints makes the overall problem **undecidable**.

Related work. For XML publishing in the pure GAV approach, and when the storage schema is purely relational, our system subsumes the expressive power of XPeranto [30] and SilkRoute [16]. For the LAV approach, we handle as particular instances XML publishing as in Agora [25] and STORED [9] and purely relational integration as in the Information Manifold [24]. The problem of rewriting regular path queries with inverse (RPQIs) with RPQI views in a LAV semistructured data context was addressed in [6, 7]. [26] gives a complete algorithm for rewriting semistructured queries with semistructured views. However, the main technical difficulties we have solved for the translation (see above) are in XQuery but not in RPQIs or the semistructured queries from [26]. While Agora captures implicitly some of the constraints inherent in the relational encoding of XML, none of the above approaches allow for *additional* constraints on the schemas. [18] and [31] propose algorithms which do take into account some constraints (e.g., referential integrity constraints) and they run in PTIME. The disadvantage here is missing rewritings (unless P=NP, since the problem is NP-hard). [17] reduces the schema correspondence given in the combined LAV and GAV approaches (GLAV) to a pure GAV correspondence. The technique does not apply to our publishing scenario because the obtained reformulation accesses *all* sources containing relevant information and thus defeats the purpose of redundant storage. [5] extends the ideas in [17] to allow for a restricted class of constraints on the published schema. [7] solves the problem for RPQI queries and RPQI views. According to our new completeness result, the C&B is a complete algorithm for minimization of (unions of) conjunctive queries under disjunctive embedded dependencies. The early work on query minimization (see [1]) did not handle dependencies. [2] lists as an open problem even the special case of the minimization of an SPJ query under functional dependencies. [19] minimizes conjunctive queries under inclusion dependencies. All of these (and more general cases) are solved by the C&B algorithm.

3 Relational Query Reformulation: The C&B Algorithm

Review: Capturing views with dependencies. The key observation that enables the uniform treatment of views and integrity constraints by the C&B algorithm is the fact that conjunctive query views can be captured by *embedded dependencies* [1] relating the input of the defining query with its output. For example, consider the view defined by

$$V(x,z) \leftarrow A(x,y), B(y,z)$$

In any instance over the schema $\{A, B, V\}$, the extent of relation V coincides with the result of this query if and only if the following dependencies hold:

$$\begin{array}{ll} (c_V) & \forall x \forall y \forall z \; [A(x,y) \wedge B(y,z) \rightarrow V(x,z)] \\ (b_V) & \forall x \forall z \; [V(x,z) \rightarrow \exists y \; A(x,y) \wedge B(y,z)] \end{array}$$

(c_V) states the inclusion of the result of the defining query in the extent of relation V, (b_V) states the opposite inclusion.

Review of C&B. Assume that in addition, the following dependency holds on the database (it is an inclusion dependency):

$$(ind) \;\; \forall x \forall y \; [A(x,y) \rightarrow \exists z \; B(y,z)]$$

Suppose that we want to reformulate the query

$$Q(x) \leftarrow A(x,y)$$

First, the query is *chased* with all available dependencies, until no more chase steps apply (see [1] for a detailed definition of the chase). The resulting query is called the *universal plan*. In our example, a chase step with (ind) yields Q_1 below, which in turn chases with (c_V) to the universal plan Q_2:

$$\begin{array}{l} Q_1(x) \leftarrow A(x,y), B(y,z) \\ Q_2(x) \leftarrow A(x,y), B(y,z), V(x,z) \end{array}$$

Notice how the chase step with (c_V) brings the view into the chase result, and how this was only possible after the chase with the semantic constraint (ind).

In the second phase of the algorithm (called the *backchase*) the *subqueries* of the universal plan are inspected and checked for equivalence with Q. Subqueries are obtained by retaining only a subset of the atoms in the body of the universal plan, using the same variables in the head. For example, $S(x) \leftarrow V(x,z)$ is a subquery of Q_2 which turns out to be equivalent to Q under the available constraints, as can be checked by chasing S "back" to Q_2 using (b_V).

A New Completeness Result. It is not accidental that we discovered a reformulation among the subqueries of the universal plan; in fact, in theorem 1 we give a theoretical guarantee that *all minimal* reformulations can be found this way. We say that a query R is *minimal under a set of constraints* C (or C-minimal) if no relational atoms can be removed from R's body, even after adding arbitrarily many equality atoms, without compromising the equivalence to R under C. Recalling the example in section 3, $T(x) \leftarrow A(x, y), V(x, z)$ is not minimal under the constraints $\{(c_V), (b_V), (ind)\}$, because we can remove the A-atom (without adding equalities) to obtain $M(x) \leftarrow V(x, z)$, which is equivalent to T, as can be checked by chasing. A query R is a *minimal reformulation* of query Q under C if it is C-minimal and equivalent to Q under C (C-equivalent to Q).

Theorem 1. *Let Q be a conjunctive query and D be a set of embedded dependencies. Assume that there is some terminating chase sequence of Q with D, yielding the universal plan U. Then any minimal reformulation of Q under D is isomorphic to a subquery of U.*

This result adds significant value to the one in [10], where we showed the completeness of the C&B when only views are allowed (i.e. we allow the constraints capturing the views such as $(c_V), (b_V)$, but no additional integrity constraints such as (ind)). Of course, checking the existence of a terminating chase sequence for a conjunctive query and arbitrary embedded dependencies is undecidable. In [10], we show that all chase sequences terminate when only the dependencies capturing the views are used. For the case of additional dependencies, we identify here a property that guarantees the termination of any chase sequence for any query.

Set of constraints with stratified-witness. Given a set C of constraints, define its *chase flow graph* $G = (V, E)$, as a directed graph whose edge labels can be either $\forall$ or $\exists$. G is constructed as follows: for every relation R of arity a mentioned in C, V contains a node R^i $(1 \leq i \leq a)$. For every pair of relations R, R' of arities a, a' and every constraint $\forall \mathbf{x}\ [\ldots \wedge R(u_1, \ldots, u_a) \wedge \ldots \rightarrow R'(v_1, \ldots v_{a'}) \ldots]$ in C, E contains the edges $(R_i, R'_j)_{1 \leq i \leq a, 1 \leq j \leq a'}$. Also, whenever the equality $x = y$ appears in the conclusion of the implication, and x, y appear as the i, j-th component of R, resp. R', E contains the edge (R_i, R'_j). Moreover, if for some j the variable v_j is existentially quantified, the edges $(R_i, R'_j)_{1 \leq i \leq a}$ are labeled with $\exists$, otherwise they are labeled with $\forall$. We say that a set of constraints has *stratified-witness* if it has no cycles through $\exists$-edges.

Denoting with $|Q|$ the size of query Q, with a the maximum arity of a relation in the schema and with l the maximum number of $\exists$-edges on a path in the chase flow graph, we have the following

Proposition 1 (with Lucian Popa). *The chase of any query Q with any set of constraints with stratified-witness terminates, and the size of the resulting query is in $O(|Q|^{a^{l+1}})$.*

This condition is efficiently checkable, and it subsumes known guarantees of the chase termination for various classes of dependencies: functional dependencies, total/full dependencies, typed 1-non-total dependencies, typed dependencies with identical sets of total attributes [3]. [6]

Remarks. 1. Notice that any pair of inclusion dependencies used to capture a view (recall $(c_V), (b_V)$ from page 6) violates the stratified-witness condition. However, the chase is guaranteed to terminate nevertheless, using the additional key observation that the introduction of the view symbol V by a chase step with (c_V) can never trigger a chase step with (b_V). This effectively breaks the $\exists$-cycle appearing in the chase flow graph.

2. When the C&B is used in the following particular scenario: (i) Q is posed against the public schema P, (ii) D gives the correspondence between P and storage schema S, and (iii) in the backchase phase we consider only subqueries expressed solely in terms of S, we obtain a complete algorithm for finding minimal reformulations.

3. By theorem 1, the C&B algorithm is a complete procedure for minimization of conjunctive queries under stratified-witness dependencies, generalizing existing procedures (see related work).

Calibrating the result. Since the backchase checks subqueries for equivalence under dependencies to the universal plan, the C&B algorithm inherits the complexity lower bounds of the equivalence check. Moreover, the C&B cannot be complete if equivalence is undecidable. A natural question is whether there are alternate algorithms that do better (are complete even when equivalence is not decidable, and have lower complexity when it is). The answer is no:

Proposition 2. *The problem of deciding minimality of a conjunctive query over all models that belong to some class C and satisfy a set of dependencies is at least as hard as deciding containment of conjunctive queries over the class C.*

In particular, the class C may be specified as all models satisfying a set of dependencies. Undecidability of containment under dependencies therefore implies that the set of minimal reformulations under dependencies is not recursive.

It turns out that the C&B algorithm is asymptotically optimal even when used as an alternative to classical algorithms for rewriting with views in the absence of additional integrity constraints (such as Minicon [28]): the associated decision problem is checking the existence of a rewriting using solely the views, in the absence of constraints. The C&B-based solution would consist in picking from the universal plan U the maximal subquery that mentions only views, and checking its equivalence to U. The complexity analysis reveals that the resulting algorithm is in NP in the size of the query, which is optimal according to [23].

Extension: DEDs. The theorem holds even when Q is a union of conjunctive queries and D is a set of *disjunctive embedded dependencies* (DEDs), as introduced in [14], which extended the chase to DEDs. Their general form is

[6] The chase flow graph is similar to the graph used to determine the existence of stratified normal forms for ILOG programs [21]. These invent object identities, just like the chase invents new variables.

$$\forall \mathbf{x}\ [\phi(\mathbf{x}) \rightarrow \bigvee_{i=1}^{l} \exists \mathbf{z_i}\ \psi_i(\mathbf{x}, \mathbf{z_i})] \tag{1}$$

where $\mathbf{x}, \mathbf{z_i}$ are tuples of variables and ϕ, ψ_i are conjunctions of *relational atoms* of the form $R(w_1, \ldots, w_l)$ and *(in)equality atoms* of the form $(w \neq w')$ $w = w'$, where $w_1, \ldots, w_l, w, w'$ are variables or constants. ϕ may be the empty conjunction. We call such dependencies *disjunctive embedded dependencies (DEDs)*, because they contain the classical embedded dependencies [1] when $l = 1$. A proper DED is (`choice`) from `TIX`.

4 XML Query Reformulation

4.1 Using relational constraints to capture XML

We treat mixed XML+relational storage uniformly by *reduction to a relational framework*. More specifically, following [12], we shall represent XML documents as relational instances [7] over the schema

$$\mathcal{X} = [\texttt{root}, \texttt{el}, \texttt{child}, \texttt{desc}, \texttt{tag}, \texttt{attr}, \texttt{id}, \texttt{text}].$$

The "intended meaning" of the relations in $\mathcal{X}$ reflects the fact that XML data is a tagged tree. The unary predicate `root` denotes the root element of the XML document, and the unary relation `el` is the set of all elements. `child` and `desc` are subsets of `el` × `el` and they say that their second component is a child, respectively a descendant of the first component. `tag` $\subseteq$ `el` × *string* associates the tag in the second component to the element in the first. `attr` $\subseteq$ `el` × *string* × *string* gives the element, attribute name and attribute value in its first, second, respectively third component. `id` $\subseteq$ *string* × `el` associates the element in the second component to a string attribute in the first that uniquely identifies it (if DTD-specified ID-type attributes exist, their values can be used for this). `text` $\subseteq$ `el` × *string* associates to the element in its first component the string in its second component.

Relational translation of XML tree navigation. Consider an XPath expression q defined as $//a$, which returns the set of nodes reachable by navigating to a descendant of the root and from there to a child tagged "a". Assume also that we materialize the view v defined as $//.//a$, i.e. which contains all "a"-children of descendants of descendants of the root. We can translate q, v as conjunctive queries Q, V over schema $\mathcal{X}$ (see [12] for details):

$$Q(y) \leftarrow \texttt{root}(r), \texttt{desc}(r, x), \texttt{child}(x, y), \texttt{tag}(y,'' a'')$$
$$V(y) \leftarrow \texttt{root}(r), \texttt{desc}(r, u), \texttt{desc}(u, x), \texttt{child}(x, y), \texttt{tag}(y,'' a'')$$

[7] We emphasize that this does not mean that the XML data is necessarily stored according to the relational schema $\mathcal{X}$. Regardless of its physical storage, we reason about XML data using $\mathcal{X}$ as its virtual relational view.

Clearly, under arbitrary interpretations of the desc relation, the two are not equivalent, and Q cannot be reformulated to use V. But on intended interpretations, the desc *relation is transitive* and therefore $R(d) \leftarrow V(d)$ is a reformulation for Q using V. Any reformulation algorithm must take into account such constraints as transitivity on the intended models of $\mathcal{X}$ lest it should miss basic reformulations.

TIX*: Constraints inherent in the XML data model.* Some (but not all!) of the intended meaning of signature $\mathcal{X}$ is captured by the set TIX (**T**rue **I**n **X**ML) of relational constraints (only the most interesting ones are listed): [8]

$$
\begin{array}{rl}
\text{(base)} & \forall x, y \; [\; \mathtt{child}(x,y) \;\; \rightarrow \;\; \mathtt{desc}(x,y) \;] \\
\text{(trans)} & \forall x, y, z \; [\; \mathtt{desc}(x,y) \wedge \mathtt{desc}(y,z) \;\; \rightarrow \;\; \mathtt{desc}(x,z) \;] \\
\text{(refl)} & \forall x \; [\; \mathtt{el}(x) \rightarrow \mathtt{desc}(x,x) \;] \\
\text{(someTag)} & \forall x \; [\; \mathtt{el}(x) \rightarrow \exists t \; \mathtt{tag}(x,t) \;] \\
\text{(oneTag)} & \forall x, t_1, t_2 \; [\; \mathtt{tag}(x,t_1) \wedge \mathtt{tag}(x,t_2) \rightarrow \;\; t_1 = t_2 \;] \\
\text{(keyId)} & \forall s, e_1, e_2 \; [\; \mathtt{id}(s,e_1) \wedge \mathtt{id}(s,e_2) \;\; \rightarrow \;\; e_1 = e_2 \;] \\
\text{(oneAttr)} & \forall x, n, v_1, v_2 \; [\; \mathtt{attr}(x,n,v_1) \wedge \mathtt{attr}(x,n,v_2) \rightarrow \;\; v_1 = v_2 \;] \\
\text{(noLoop)} & \forall x, y \; [\; \mathtt{desc}(x,y) \wedge \mathtt{desc}(y,x) \rightarrow \;\; x = y \;] \\
\text{(oneParent)} & \forall x, y, z \; [\; \mathtt{child}(x,z) \wedge \mathtt{child}(y,z) \rightarrow \;\; x = y \;] \\
\text{(oneRoot)} & \forall x, y \; [\; \mathtt{root}(x) \wedge \mathtt{root}(y) \rightarrow x = y] \\
\text{(topRoot)} & \forall x, y \; [\; \mathtt{desc}(x,y) \wedge \mathtt{root}(y) \rightarrow \mathtt{root}(x)] \\
(\mathtt{line}) & \forall x, y, u \; [\; \mathtt{desc}(x,u) \wedge \mathtt{desc}(y,u) \rightarrow x = y \vee \mathtt{desc}(x,y) \vee \mathtt{desc}(y,x)] \\
\text{(choice)} & \forall x, y, z \; [\; \mathtt{child}(x,y) \wedge \mathtt{desc}(x,z) \wedge \mathtt{desc}(z,y) \rightarrow x = z \vee y = z]
\end{array}
$$

Note that these axioms are First-Order incomplete; they don't even prove $\forall x \forall y \; \mathtt{desc}(x,y) \rightarrow x = y \vee \exists z \; \mathtt{child}(x,z) \wedge \mathtt{desc}(z,y)$. Still they are special because they are sufficient to give an optimal, chase-based decision procedure for containment of XQueries from the fragment with NP-complete containment [12].

Notice that except for (line) and (choice), all constraints in TIX are *embedded dependencies* (as [1] calls them, but also known as tuple- and equality-generating dependencies [3]) for which a deep and rich theory has been developed. (line) contains disjunction but so do XQueries. Extending the theory to *disjunctive embedded dependencies* is fairly straightforward [14].

Transitive Closure and Treeness. Observe that (base), (trans), (refl) above only guarantee that desc *contains* its intended interpretation, namely the reflexive, transitive closure of the child relation. There are many models satisfying these constraints, in which desc is interpreted as a proper superset of its intended interpretation, and it is well-known that we have no way of ruling them

[8] A collection $D_1, \ldots, D_n$ of XML documents is represented by the disjoint union of schemas $\mathcal{X}_i$ and the union of constraints in each $\mathtt{TIX}_i$, where each $\mathcal{X}_i$ ($\mathtt{TIX}_i$) is obtained from $\mathcal{X}$ (resp. TIX) by subscripting all relational symbols with i.

out using first-order constraints, because transitive closure is not first-order definable. Similarly, the "treeness" property of the `child` relation cannot be captured in first-order logic. The fact that we can nevertheless decide equivalence of behaved XQueries (containing descendant navigation) over the intended interpretation using the constraints in `TIX` and classical relational (hence first-order) techniques comes therefore as a pleasant surprise.

4.2 XML Queries and Constraints

According to their operational semantics, XQueries compute in two phases. First, the navigation part of an XQuery searches the input XML tree(s) binding the query variables to nodes or string values. In a second phase that uses the tagging template a new element of the output tree is created for each tuple of bindings produced in the first phase (see example 1 below).

Describing the navigational part: decorrelated XBind queries. In this paper, we focus on reformulating the navigational part of a client XQuery. In order to describe it, we introduce a simplified syntax that disregards the element construction, focusing only on binding variables and returning them. We call the queries in this syntax XBind queries. Their general form is akin to conjunctive queries. Their head returns a tuple of variables, and the body atoms can be purely relational or are predicates defined by XPath expressions with restrictions (see [12, 13] for their syntax). The predicates can be binary, of the form $[p](x, y)$, being satisfied whenever y belongs to the set of nodes reachable from node x along the path p. Alternatively, predicates are unary, of form $[p](y)$, if p is an absolute path starting at the root.

Example 1. Consider the query Q defined as

```
for $a in distinct(//author/text()) return
    <item> <writer>$a</writer>
              {for   $b in //book, $a1 in $b/author/text(), $t in $b/title
              where  $a = $a1 return $t}
    </item>
```

Note the nested query shown in braces, which is correlated with the outer one through free variable \$a. It returns *copies* of the `title` subelements of books whose author \$a1 coincides with \$a. The direct, nested loop-based evaluation of Q is inefficient. Research in evaluating correlated SQL queries suggests an alternative strategy that consists in breaking the query into two decorrelated queries which can be efficiently evaluated separately and then putting together their results using an outer join [29]. We will borrow this technique, obtaining for Q the two decorrelated XBind queries below (\$ signs are dropped from the variable names). Xb_o (Xb_i) computes the bindings for the variables introduced in the outer (inner) `for` loop. Notice that Xb_i also outputs the value of free variable \$a in order to preserve the correlation between bindings.

$$Xb_o(a) \leftarrow [//author/text()](a)$$

$$Xb_i(a, b, a1, t) \leftarrow Xb_o(a), [//book](b), [./author/text()](b, a1), [./title](b, t), a = a1$$

In summary, we describe the navigational part of an XQuery by a set of decorrelated XBind queries. Using the translation of XPath expressions to conjunctions of relational atoms from signature $\mathcal{X}$ (sketched on page 9 and detailed in [12]), we obtain a straightforward translation of any XBind query to a union of conjunctive queries.

XML Integrity Constraints (`XICs`). In [12], we use the same syntax for predicates defined by XPath expressions to define a class of XML integrity constraints (`XICs`). It turns out that `XICs` are related to XBind queries in the same way in which embedded dependencies are related to conjunctive queries: implication and containment are inter-reducible (Proposition 3). `XICs` have the same general form as (1), but the relational atoms are replaced by predicates defined by restricted XPath expressions, just like for XBind queries (see (2) below). Using the same translation of XPath expressions as for XBind queries, we get a straightforward translation of `XICs` to DEDs.

$$\forall x, y\ [/pers](x) \wedge [.//dog](x, y) \to \exists z\ [./pets](x, z) \wedge [.//](z, y) \tag{2}$$

Behaved XQueries. The strategy of our algorithm is to compile each XBind query to a union of conjunctive queries, compile all XQuery views (not just their XBind parts; tagging part as well!) to DEDs and apply the C&Bs algorithm to the relational problem. There are of course XQuery features we cannot compile to dependencies. User-defined functions, aggregates and universally quantified path qualifiers [33] are the main examples. We emphasize that the soundness of the algorithm presented below holds for any query that is compilable relationally. However, its completeness is guaranteed only for a restricted class of XQueries, which we call behaved. In addition to ruling the non-compilable features out, behaved XQueries satisfy a few more restrictions. [9] The main restriction rules out navigation to parent and wildcard child (i.e. child of unspecified tag) (more on this counterintuitive restriction shortly). This class is still quite expressive: it allows navigation to ancestor,descendant and child of *specified* tag; disjunction and path alternation; inequalities; equalities on values (text and attributes) and on node identities. The query in example 1 is behaved. From a practical perspective, the features that we cover are in our experience the most common ones anyway, with the exception of aggregates. As discussed shortly, even modest relaxation of these restrictions results in incompleteness of reformulation, suggesting that different techniques are needed beyond this class of XQueries.

4.3 Compiling Schema Correspondences

Obstacles in capturing XQuery views with dependencies. In section 3, we show how we express a conjunctive query view using two inclusion dependencies. This technique does not apply directly to XQuery views, which are more expressive:

[9] See [13, 11] for the detailed description of this class of XQueries, or [12] for the behaved fragment of XPath used in behaved XQueries.

(i) XQueries contain nested, correlated subqueries in the return clause, (ii) they create new nodes, which do not exist in the input document, so there is no inclusion relationship between input and output node id sets, and (iii) they return deep, recursive copies of elements from the input. We sketch the solution using example 1 (see [11, 13] for more details).

Nested, Correlated Subqueries. Recall that the navigation part of an XQuery is described by a set of decorrelated XBind queries (Xb_o,Xb_i in example 1). Also recall that every XBind query can be straightforwardly translated to a union of conjunctive queries over schema $\mathcal{X}$. This union can now be captured with two DEDs, as shown on page 6.

Construction of New Elements. For every binding for \$a, a new `item` element node is created whose identity does not exist anywhere in the input document, but rather is an invented value. Distinct bindings of \$a result in distinct invented `item` elements. In other words, the identities of the `item` element nodes are the image of the bindings for \$a under some injective function F_{item}. [10] We capture this function by extending the schema with the relational symbol G_{item}, intended as the graph of F_{item} ($G_{item}(x, y) \Leftrightarrow y = F_{item}(x)$) and use dependencies to enforce this intended meaning.

Deep Copies of Elements. Here is how we capture the fact that Q returns, for every binding of \$a, a copy of the tree rooted at the `title`-element node which \$t was bound to. We model copying by an injective function F^a_t which, for a fixed \$a, takes as argument any node n in the tree rooted at \$t, and outputs an invented node n' that is a copy of n. We say that n' is an (\$a,\$t)-copy of n to emphasize that there is one copy of the tree rooted at \$t for each value of \$a. We represent the family of (\$a,\$t)-copy functions $\{F^a_t\}_{a,t}$ by the relation C: $\forall a \forall t \; F^a_t(n, n') \Leftrightarrow C(a, t, n, n')$. Again, we capture the intended meaning for C using DEDs. The sample DED (3) states that if n' is an (\$a,\$t)-copy of n, then the descendants of n are (\$a,\$t)-copied as descendants of n' (Q's output is encoded as an instance over schema $\mathcal{X}_2$):

$$\forall a \forall t \forall n \forall n' \forall d \; [C(a, t, n, n') \wedge \mathtt{desc}_1(n, d) \rightarrow \exists d' \; \mathtt{desc}_2(n', d') \wedge C(a, t, d, d')] \quad (3)$$

Compiling relational-to-XML encodings. Recall from Step 1 in section 1 that in order to uniformly express the schema mappings as XQueries, we encode the relational data as XML. Various schemes have been proposed, all having in common the fact that each relational tuple corresponds to an XML subtree whose nodes have fresh, invented identities. We have encountered a similar situation when compiling XQuery views, where the tuples were the variable bindings produced by the XBind queries and the XML trees were given by the element constructor. We therefore use similar DEDs to capture the encoding.

[10] Many semistructured and XML query languages use functions like F_{item} as explicit query primitives, under the name of Skolem functions. Our technique for compiling into dependencies fits seamlessly with an extension of XQuery with Skolem functions.

4.4 The Algorithm

If any variables of the XBind query Xb are bound to element nodes, then Xb cannot be reformulated against the storage schema: if the latter is relational, it contains no XML nodes, and if it is mixed, then the node identities in the storage and published data are disjoint. We hence need to find query "plans" which collect data from the storage but also *invent* and *copy* nodes, according to the semantics of the XQuery views that define the schema correspondence.

Plans: reformulations using auxiliary schema. We show in section 4.3 how to model this semantics using Skolem and copy functions. Suppose a plan retrieves the storage data tuples that satisfy condition $c(\mathbf{x})$ and returns $\mathbf{y}$ and an invented node $n = F(\mathbf{z})$ where F is a Skolem function and $\mathbf{y}, \mathbf{z} \subseteq \mathbf{x}$. This plan can be described as the query $P(\mathbf{y}, n) \leftarrow c(\mathbf{x}), G(n, \mathbf{z})$, with G the graph of F ($G(n, \mathbf{z}) \Leftrightarrow n = F(\mathbf{z})$). Denote with `Aux` the relational symbols modeling the graphs of Skolem and copy functions. Then any plan can be represented by a query against the extended storage schema $S \cup \mathtt{Aux}$.

Algorithm **for XBind reformulation.**
Given:

– an XBind query Xb (obtained from a behaved XQuery)

– a schema correspondence described by a set of behaved XQuery views $\mathbf{V}$.

– the set $\mathbf{C_X}$ of `XIC`s over the various XML documents (public or storage)

– the set $\mathbf{C_R}$ of relational integrity constraints over the relational part of the storage schema S.

Do:

– Compile Xb to the union of conjunctive queries $c(Xb)$

– Compile the schema correspondence to the set of DEDs $c(\mathbf{V})$. In the process, we introduce the set `Aux` of Skolem and copy function graphs (see section 4.3).

– Compile $\mathbf{C_X}$ to the set $c(\mathbf{C_X})$ of DEDs.

– Let $\mathbf{R}$ be the set of reformulations *against* $S \cup \mathtt{Aux}$ obtained by applying the C&B algorithm to $c(Xb)$ under $\mathtt{TIX} \cup c(\mathbf{V}) \cup c(\mathbf{C_X}) \cup \mathbf{C_R}$.

Return:

– all queries in $\mathbf{R}$ that correspond to a viable reformulation plan. **End.**

Bounded `XIC`s. In [12], we introduce the class of *bounded* `XIC`s, such that we can guarantee the termination of the chase with $c(C_{\mathbf{X}})$. We also show there that containment of XBind queries is undecidable in the presence of `XIC`s that make even modest use of unboundedness. From proposition 2 it follows that no minimization algorithm is complete for unbounded `XIC`s. A bounded `XIC` allows existential quantification over element nodes, but only when their depth in the XML tree is bounded by the size of the `XIC`. The class is quite expressive, it contains XML Schema key constraints, many keyref constraints, and constraints implied by the content model definition of XML elements. In section 4.2, the `XIC` (2) is bounded, but $\forall x\ [//\mathit{employee}](x) \rightarrow \exists y\ [//\mathit{employer}](y)$ is not.

Theorem 2 (Relative Completeness). *If the constraints in* $\mathbf{C_X}$ *are bounded and* $\mathbf{C_R}$ *has stratified-witness, then* R *is a minimal reformulation of* Xb *if and only if* $c(R)$ *is a minimal reformulation of* $c(Xb)$ *under* $\mathtt{TIX} \cup c(\mathbf{V}) \cup c(\mathbf{C_X}) \cup \mathbf{C_R}$.

Theorems 2 and 1 immediately imply the following

Corollary 1 (Overall Completeness). *The algorithm finds all minimal reformulatons for behaved client XBind queries, under behaved XQuery views, bounded* `XIC`*s and stratified-witness relational dependencies.*

Remark. It follows from proposition 2 that even modest use of non-behaved features such as wildcard child navigation results in an incomplete algorithm unless $NP = \Pi_2^p$: in [12] we show that containment for XBind queries with wildcard child is Π_2^p-hard even when the queries are disjunction-free and use no ancestor navigation. On the other hand, it turns out that the C&B gives us a reformulation in NP in the size of these queries.

5 Constraint Reformulation

We present a way to reuse any query reformulation algorithm for constraint reformulation, exploiting the following fundamental reduction between query containment and constraint satisfaction.

Proposition 3. *(a) For every* `XIC` *d there are XBind queries Q_1^d, Q_2^d such that for any instance I, $I \models d \Leftrightarrow Q_1^d(I) \subseteq Q_2^d(I)$. (b) For every XBind queries Q_1, Q_2, there is an* `XIC` *$cont(Q_1, Q_2)$ such that for every instance I, $Q_1(I) \subseteq Q_2(I) \Leftrightarrow I \models cont(Q_1, Q_2)$.*

Proof: (a) For d of form $\forall \mathbf{x}\ [B(\mathbf{x}) \rightarrow \exists \mathbf{y}\ C(\mathbf{x}, \mathbf{y})]$, construct $Q_1^d(\mathbf{x}) \leftarrow B(\mathbf{x})$ and $Q_2^d(\mathbf{x}) \leftarrow B(\mathbf{x}) \wedge C(\mathbf{x}, \mathbf{y})$. (b) For $Q_1(\mathbf{x}) \leftarrow B_1(\mathbf{x}, \mathbf{y})$ and $Q_2(\mathbf{x}) \leftarrow B_2(\mathbf{x}, \mathbf{z})$, $cont(Q_1, Q_2) = \forall \mathbf{x} \forall \mathbf{y}\ [B_1(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ B_2(\mathbf{x}, \mathbf{z})]$.

`XIC` **reformulation algorithm:** (1) construct Q_1^d, Q_2^d, (2) reformulate each against $S \cup$ `Aux`, to R_1, resp. R_2, (3) construct $cont(R_1, R_2)$, and (4) return the restriction of $cont(R_1, R_2)$ to S. •

Since in general d quantifies over XML nodes (recall `XIC` (2) in section 4.2), Q_1^d, Q_2^d cannot be reformulated against the storage schema S only, as it does not contain these nodes. However, Q_1^d, Q_2^d are XBind queries, which we can reformulate against $S \cup$ `Aux`. By the following result, we can always turn $cont(R_1, R_2)$ (against $S \cup$ `Aux`) into a dependency formulated solely against S:

Proposition 4. *Let d_R be obtained from $cont(R_1, R_2)$ by simply dropping all atoms involving any variable x appearing as the result of a function from* `Aux`*. Then on all instances satisfying the schema correspondence, $cont(R_1, R_2)$ is satisfied if and only if d_R is.*

Notice that by "plugging in" any sound query reformulation algorithm, we obtain a sound algorithm for constraint reformulation. Details are provided in [13, 11].

6 Summary

We have presented an algorithm for finding the minimal reformulations of client XQueries in XML publishing scenarios, when the correspondence between public and storage schema is given by a combination of GAV and LAV XQuery views. The algorithm handles in the same unified way redundant storage (typical in XML applications), constraints in XML data (as specified by XML Schema) and constraints in the relational storage. The algorithm is complete and asymptotically optimal for an expressive class of client query and views (behaved XQueries) and integrity constraints (bounded `XIC`s and stratified-witness DEDs). The algorithm can be reused for reformulation of `XIC`s. Given its direction-independence, it applies also to reformulating integrity constraints on the storage to constraints on the public schema. This is useful for publishing integrity constraints to help clients understand the semantics of the published data.

Practicality of the approach. We have built a query reformulation system [11] based on the method presented here. Putting these ideas to work required a good deal of challenging engineering but, as we plan to report elsewhere, the performance of the resulting system proves that the method is definitely practical.

Acknowledgements We are very grateful to Lucian Popa for his contributions. We also thank Dan Suciu, Mary Fernandez, Susan Davidson, Peter Buneman, Yi Chen and Yifeng Zheng for their useful suggestions.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2. A. Aho, Y. Sagiv, and J. Ullman. Efficient optimization of a class of relational expressions. In *TODS*, 4(4), 1979.
3. C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *JACM*, 31(4):718–741, 1984.
4. P. Buneman, S. Davidson, W. Fan, C, Hara, and W.-C. Tan. Keys for xml. In *WWW10*, May 2001.
5. A. Cali, G. De Giacomo, and M. Lenzerini. Models of information integration: Turning local-as-view into global-as-view. In *FMII*, 2001.
6. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. In *PODS*, 1999.
7. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. View-based query processing for regular path queries with inverse. In *PODS*, 2000.
8. M. Carey, J. Kiernan, J. Shanmugasundaram, E. Shekita, and S. Subramanian. XPERANTO: Middleware For Publishing Object-Relational Data as XML Documents. In *VLDB*, Sep 2000.
9. A. Deutsch, M. F. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. In *SIGMOD*, 1999.
10. A. Deutsch, L. Popa, and V. Tannen. Physical Data Independence, Constraints and Optimization with Universal Plans. In *VLDB*, 1999.

11. A. Deutsch. XML Query Reformulation Over Mixed and Redundant Storage. *PhD thesis*, University of Pennsylvania, 2002. Available from `http://db.cis.upenn.edu/cgi-bin/Person.perl?adeutsch`
12. A. Deutsch and V. Tannen. Containment and Integrity Constraints for XPath Fragments. In *KRDB*, 2001.
13. A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints (extended version). Available from `http://db.cis.upenn.edu/cgi-bin/Person.perl?adeutsch`
14. A. Deutsch and V. Tannen. Optimization Properties for Classes of Conjunctive Regular Path Queries. In *DBPL*, 2001.
15. M. Fernandez, A. Morishima, and D. Suciu. Efficient Evaluation of XML Middleware Queries. In *SIGMOD*, 2001.
16. M. Fernandez, W. Tan, and D. Suciu. SilkRoute: Trading between Relations and XML. In *WWW9*, 2000.
17. M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *AAAI/IAAI*, 1999.
18. J. Goldstein and P. A. Larson. Optimizing queries using materialized views. In *SIGMOD*, 2001.
19. J. Gryz. Query folding with inclusion dependencies. In *ICDE*, 1998.
20. A. Halevy. Logic-based techniques in data integration. In *Logic Based Artificial Intelligence*, 2000.
21. R. Hull and M. Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers. In *VLDB*, 1990.
22. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.
23. A. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, 1995.
24. A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, 1996.
25. I. Manolescu, D. Florescu, and D. Kossman. Answering XML Queries on Heterogeneous Data Sources. In *VLDB*, 2001.
26. Y. Papakonstantinou and V. Vassalos. Query Rewriting for Semistructured Data, In *SIGMOD*, 1999.
27. L. Popa. *Object/Relational Query Optimization with Chase and Backchase*. PhD thesis, Univ. of Pennsylvania, 2000.
28. R. Pottinger and A. Halevy. Minicon: A scalable algorithm for answering queries using views. In *VLDB Journal*, 10(2-3), 2001.
29. P. Seshadri, H. Pirahesh, and T. Y. C. Leung. Complex query decorrelation. In *ICDE*, 1996.
30. J. Shanmugasundaram, J. Kiernan, E. J. Shekita, C. Fan, and J. Funderburk. Querying XML Views of Relational Data. In *VLDB*, 2001.
31. O. Tsatalos, M. Solomon, and Y. Ioannidis. The gmap: A versatile tool for physical data independence. *VLDB*, 1994.
32. W3C. XML Schema Part 0: Primer. Working Draft 25 February 2000. Available from `http://www.w3.org/TR/xmlschema-0`.
33. W3C. XQuery: A query Language for XML. W3C Working Draft 15 February 2001. Available from `http://www.w3.org/TR/xquery`.