



XML queries and constraints, containment and reformulation

Alin Deutsch^{a,*}, Val Tannen^b

^a*Department of Computer Science and Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA*

^b*Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd Street, Philadelphia, PA 19104, USA*

Abstract

Starting from the XQuery language we define XBind, an XML analog of relational conjunctive queries as well as a related class of XML integrity constraints (dependencies). We identify a fragment of XBind for which containment is decidable, in fact Π_2^P -complete, and a further fragment for which containment is NP-complete. We extend the containment algorithm to take XML dependencies into account. We give an algorithm for the reformulation of XBind queries under combinations of GAV and LAV XQuery views, as well as additional dependencies. 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. Our results on containment and reformulation depend on certain restrictions on the query and constraint languages. We calibrate the results by showing that lifting these restrictions significantly changes the complexity of the problems.

© 2004 Elsevier B.V. All rights reserved.

1. Introduction

For the relational data model there exists a rich and interesting theory of conjunctive queries and of the (embedded) dependencies¹ corresponding to them. Query containment

* Corresponding author, supported in part by NSF/CAREER grant 0347968.

E-mail addresses: deutsch@cs.ucsd.edu (A. Deutsch), val@cis.upenn.edu (V. Tannen).

¹ Also known as *integrity constraints*.

and minimization, plain or under dependencies, were studied during the classical times (see [1] for references). Query reformulation, specifically rewriting with views, was studied more recently (see the survey [18]). A lot of interest in a similar theory for the XML data model is now emerging and this paper is an attempt to contribute to such a theory.

In fact, our strategy is to solve the XML query containment and reformulation problems via sound and complete reductions to relational problems that can be solved with the *chase* technique and with the *Chase&Backchase* (C&B) algorithm.² The querying instruments that have been standardized for XML, especially XQuery [33] and its critical component XPath [31], are quite expressive and have been designed to deal with non-first-order features like transitive closure and tree data. In view of this, the large size of the XPath/XQuery fragments for which our relational reduction works is a pleasant surprise.

The first problem is to identify within XML query and constraint formalisms the fragments that can be the analogs of the relational queries and dependencies with a nice theory. We begin by looking at the semantics of an XQuery example.

Example 1.1. Consider an XML document containing book elements, each of which contains a title and some author subelements. The query Q below restructures the data by grouping the book titles with each author. The groups appear as item elements, whose writer subelement contains the author name and whose (possibly multiple) title subelements contain all titles (co-) authored by this writer.

```
Q:  <result>
      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>
    </result>
```

XQuery relies on XPath expressions such as `//author/text()` to navigate through the input document. XPath pattern-matching binds the XQuery variables to the document's elements, text values, etc. In fact, Q 's computation can be described in two stages. First, all bindings for the variable $\$a$ to distinct text values of author elements are computed. Next, a unique result root element is created and for every binding of $\$a$, a new item subelement of this result element is created. Each item element has a writer subelement containing the text $\$a$ was bound to, and as many title subelements as are returned by the nested inner query shown in braces. Notice that the inner query is correlated with the outer query through the occurrence of the variable $\$a$. The execution of the inner query is also in two stages. First a set of triples of bindings for the triple of variables $\$b$, $\$a1$, $\$t$ is computed (of course, the

² We introduced C&B with Lucian Popa in [8]. In [13] we proved that C&B is complete for relational minimization under dependencies.

binding of $\$a1$ is always the same as the current binding of $\$a$). In the second stage, for each triple of bindings the inner query returns a *copy* of the element bound to $\$t$.

The first stage of the XQuery semantics is reminiscent of the evaluation of relational conjunctive queries. The analogy is strengthened by the fact that the semantics of XPath expressions [34] consists of unary or binary relations over element (node) identities and/or strings. We are naturally led to a syntax like that of conjunctive queries, but with atoms defined by XPath expressions (in addition to usual relation predicates). For instance, we associate to the query Q in Example 1.1 the following queries:

$$\begin{aligned} Xb_o(a) &\leftarrow \text{[//author/text()]}(a), \\ Xb_i(a, b, a1, t) &\leftarrow Xb_o(a), \text{[//book]}(b), \text{[./author/text()]}(b, a1), \text{[./title]}(b, t), a = a1. \end{aligned}$$

The XPath atoms are understood as relations. For example, $\text{[./author/text()]}(b, \$a1)$ is true iff $a1$ is the text inside an element (node) tagged *author* who is a child of the node b . And $\text{[//book]}(b)$ is true iff b is a child tagged *book* of some element that is a descendant of the root (in fact, all nodes are descendants of the root). The rest of the semantics is as for conjunctive queries. Hence, Xb_o computes the bindings for the outer query, while Xb_i computes the bindings for the inner query, for each a in the outer query. Notice that a is also in Xb_i 's output, in order to preserve the correlation between variable bindings.

We call such queries *XBind* queries because they fully capture the first stage of XQuery evaluation in which the document is navigated, patterns are matched, and all the bindings for the variables are computed. XBind queries play for us the role of conjunctive queries, with some restrictions on the XPath expressions used, as we shall see.

Note that relational conjunctive queries (for binary relation schemas) can be seen straightforwardly as particular cases of XBind queries. But the semantics of XBind queries is more complicated, with more containments/equivalences holding, e.g.:

$$\begin{aligned} C(x) \leftarrow \text{[a]}(x) \text{ is contained in } D(x) \leftarrow \text{[//a]}(x), \\ E(x) \leftarrow \text{[a/b]}(x), \text{[./b]}(y, x), \text{[./c]}(y, z) \text{ is equivalent to } F(x) \leftarrow \text{[a[c/b]}(x). \end{aligned}$$

A central concern of this paper is the problem of *reformulation* of XQueries. 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 (see Fig. 1). 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. An application concerning specifically XQuery is XML *publishing*, where P is the public XML schema while S is the storage schema of proprietary data, which can be a mixture of native XML repositories and relational DBMSs. The public data is virtual, hence an XQuery Q formulated against P must be reformulated as a query X that can be actually executed on the stored proprietary data.

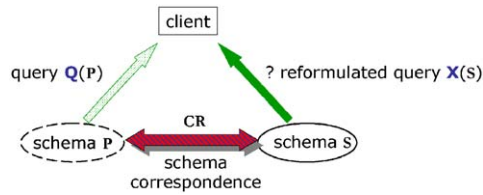


Fig. 1. General problem of query reformulation.

A important issue is how to model the schema correspondence CR . Two main approaches have been used for this. With the terminology used in data integration [18,21] we have “global-as-view” (GAV) and “local-as-view” (LAV) with the views themselves (sometimes called *mappings*) expressed in a query language and directed as follows:

$$\text{GAV} : S \rightarrow P, \quad \text{LAV} : P \rightarrow S.$$

Assuming that we know how to compose queries and views, we have

$$Q \circ \text{GAV} = X, \quad X \circ \text{LAV} = Q.$$

We see that GAV reformulation amounts to, and is called *composition-with-views* while LAV reformulation amounts to “solving an equation”, which is quite a bit harder, may have multiple solutions, and is often called *rewriting-with-views*.

In fact, our approach allows *both* GAV and LAV views in the schema correspondence, each of them a mapping from a portion of S to a portion of P or conversely. This is very useful in general and is actually crucial in XML publishing: GAV views are used for hiding portions of the proprietary data, while LAV views are used to describe redundant stored data such as materialized views or cached query answers, which play an important role in tuning the performance of applications.

Moreover, we shall assume that the views, GAV or LAV, are expressed in XQuery. In this we agree with [4] that stored relational data can be easily understood through virtual XML encodings thus facilitating design and administration tasks when the data is mixed. Although the query to be reformulated is expressed in XQuery, it turns out that only the navigation/variable binding part depends on the schema correspondence. As in [4,17,24], we split off the navigation part of an XQuery and therefore we concentrate only on the *reformulation of XBind queries*. At the same time, for the views we cannot make this simplification since their output is essential in reformulation.

In many applications containment and reformulation are considered only over classes of documents that satisfy certain *integrity constraints*. While much is known about relational constraints, XML constraint formalisms are still “under construction” so we allowed ourselves to be closely guided by an analogy with the relational case when defining a class of dependencies for XML. Just as relational (embedded) dependencies [1], also called tuple- and equality-generating dependencies (tgd’s and egd’s) in [2], correspond closely to relational conjunctive queries, we define **XML Integrity Constraints (XIC)** to relate closely to XBind queries. Hence we use conjunctions of atoms defined by XPath expressions as in

XBind and we use the same logical format (quantifiers and implication) as in the relational case.

Example 1.2. For an XML document recording persons, their addresses as children nodes, and their social security numbers (ssn) as an attribute, (1), (2) and (3) below state respectively that persons have at most one address, at least one address, and that the `ssn` attribute is a key for person elements:

$$\forall p \forall a_1 \forall a_2 [//person](p) \wedge [./address](p, a_1) \wedge [./address](p, a_2) \rightarrow a_1 = a_2, \quad (1)$$

$$\forall p [//person](p) \rightarrow \exists a [./address](p, a), \quad (2)$$

$$\forall x, y, s [//person](x) \wedge [//person](y) \wedge [./@ssn](x, s) \wedge [./@ssn](y, s) \rightarrow x=y. \quad (3)$$

In general, XICs are expressive enough to capture a considerable part of XML Schema [3,32] including keys and “keyrefs”. Expressiveness is both good and bad: XICs include as particular cases the relational dependencies (tgd’s and egd’s for binary relation schemas, that are enough to axiomatize undecidable theories [1]. Moreover, XICs have their own sources of complexity, also leading to undecidability (see “(un)boundedness” below).

We can now state the main concerns of this paper:

- Containment of XBind queries under XICs,
- reformulation of XBind queries under GAV and LAV XQuery views, and XICs,
- calibration of the restrictions. (Our results on containment and reformulation depend on certain restrictions on XPath, XQuery, and XICs.)

The results presented in this journal paper have already been announced in our conference papers [13] and [9]. Here we give a more detailed presentation that contains most of the proofs. In Section 2 we define the fragments of query and constraint languages for which we can prove our decidability and completeness results. In Section 3 we describe the basics of the translation from XML to relations that underlies our approach. In Section 4 we give our decidability and complexity characterization results for containment of XBind queries, alone and under XICs. In Section 5 we give the translation of XQuery views into relational constraints, the reformulation algorithm, and its completeness property. In Section 6 we show that lifting the restriction on the queries and constraints that we used in our results does indeed change significantly the nature of the problems. We end with related and future work.

2. XML queries and constraints

We define here the fragments of XPath (hence XBind and XIC) and of XQuery on which we can apply our proof techniques.

2.1. AT-XPath, LAT-XPath

The **all-tagged** fragment of XPath (AT-XPath) is defined by the following grammar (based on the grammar and semantics given in [34]).

```

    xpath ::= sep p | . sep p
(separator) sep ::= / | //
    (path) p ::= p1|p2 | p1 sep p2 | p[q] | . | n | @n | @ * | text() | ancestor-or-self | id(p) | id(s)
    (qualifier) q ::= q1 and q2 | q1 or q2 | p | p = s | p1 = p2 | p1 == p2 | p ≠ s | p1 ≠ p2 | p1 ≠= p2

```

Above, n is any tag or attribute name, and s is any string constant. $==$ stands for equality on node identities, and $id(p)$ returns the set of nodes whose ID attributes are the set of strings returned by path p . Of course, as in conjunctive queries, arbitrary negation is ruled out. A non-intuitive but salient restriction is the absence of navigation to children of an unspecified tag (wildcard child), hence the name of the fragment: all-tagged. More on ruling out wildcard child below. Parent navigation is ruled out because, together with equality on node identities, it can express wildcard child navigation. Proper ancestor navigation is also ruled out, as it contains at least one parent navigation step. Minor restrictions include ruling out the following/preceding navigation axes (handling document order is a separate, challenging research issue), and universal quantification in the qualifiers (this, together with non-equality, allows us to express set inclusion and difference, which makes the containment problem undecidable). Observe that the *AT-XPath* fragment is still quite expressive: it allows navigation to descendant-or-self/ancestor-or-self, arbitrary equalities (on values: text, attributes; on node identities) disjunction/alternation, limited negation (non-equalities).

A further restriction is the **linear** fragment of *AT-XPath* which we denote *LAT-XPath*. This fragment is obtained by disallowing path alternation, qualifier disjunction, ancestor-or-self navigation, non-equalities and the equality on node identities ($==$).

2.2. AT-XQuery

Views will be described in a fragment of XQuery that we also call *all-tagged* and denote it *AT-XQuery*. It is defined with following grammar rules, in addition to the ones above (here v is any variable name, and s is any string constant):

```

    query ::= for bindings where conditions return output
    bindings ::= binding | binding , bindings
    binding ::= var in path
    path ::= var sep p | document( s ) sep p
(variable) var ::= $ v
    conditions ::= condition | condition and conditions | condition or conditions
    condition ::= some var in path satisfies condition
                | path = s | path1 = path2 | path1 == path2
                | path ≠ s | path1 ≠ path2 | path1 ≠= path2
    output ::= content | template
    content ::= var | s | { query }
    template ::= ⟨ n ⟩ content ⟨ / n ⟩ | ⟨ n ⟩ template ⟨ / n ⟩
                | template1 template2

```

The major restriction here is the absence of aggregates. As before, arbitrary negation and even universal quantification are disallowed in the conditions. A minor restriction is the absence of user-defined functions (which take the semantics into uncharted territory). The query in Example 1.1 belongs to *AT-XQuery*. 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 in Section 6 even modest relaxation of these restrictions changes the complexity of the problems, suggesting that different techniques are needed beyond this class of XQueries.

2.3. XBind queries

Like [4,24] we follow [17] in splitting **XQuery** = **navigation part** + **tagging template** corresponding to the two phases in the operational semantics of XQuery [33]. 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.³ The first phase can be captured by a simplified syntax that disregards the element construction, focusing only on the binding of query variables. 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 [31]. 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 $[p](y)$, whenever p is an absolute path starting from the root (recall queries Xb_o , Xb_i from Example 1.1).

2.4. AT-XBind, LAT-XBind

When the XPath expressions used in XBind are in AT-XPath we denote the corresponding class of XBind queries by AT-XBind. The restriction that corresponds to LAT-XPath is a little more complicated, because we want it to also correspond to the navigation part of certain XQueries. These XQueries are not only restricted to using LAT-XPath but their `where` clause cannot have disjunction, equality on node identities and non-equalities. A careful analysis of these restrictions produces the following definition. A LAT-XBind query is an XBind query that uses only LAT-XPath and moreover is such that variables that bind to element nodes (as opposed to text and attribute values) may appear no more than once in a unary atom or in the second position of a binary atom.

2.5. Review: disjunctive embedded dependencies (DEDs)

We recall the definition of DEDs from [11]. These are first-order logic assertions of the form

$$\forall \mathbf{x} \left[\phi(\mathbf{x}) \rightarrow \bigvee_{i=1}^l \exists \mathbf{z}_i \psi_i(\mathbf{x}, \mathbf{z}_i) \right], \quad (4)$$

where \mathbf{x} , \mathbf{z}_i are tuples of variables and ϕ , ψ_i are conjunctions of *relational atoms* of the form $R(w_1, \dots, w_l)$ and *(dis)equality atoms* of the form $(w \neq w')$ $w = w'$, where w_1, \dots, w_l, w, w' are variables or constants. ϕ may be the empty conjunction. The definition of DEDs contains for $l = 1$ that of the classical embedded dependencies [1] (as [1] calls them, but also known

³ Previous research has addressed the efficient implementation of the second phase [16,30].

as tgd 's and egd 's [2]) for which a deep and rich theory has been developed. Extending the theory to DEDs was suggested already in [2] and is fairly straightforward. The main difference to the classical chase is that, instead of a chase *sequence*, the rewrite yields a chase *tree*, whose leaves are conjunctive queries to which no more chase step applies (details are in [7,10,11] and in Appendix A).

2.6. Set of constraints with stratified-witness

In general, checking the termination of the chase with embedded dependencies (and hence DEDs) is undecidable. We identify here a sufficient condition that guarantees termination of any chase sequence with DEDs. This condition is efficiently checkable, and it subsumes previously 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 [2] and sets of acyclic inclusion dependencies [6].

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 $\forall \mathbf{x} [\dots \wedge R(u_1, \dots, u_a) \wedge \dots \rightarrow \dots R'(v_1, \dots, v_{a'}) \dots]$ 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 .⁴ We say that a set of constraints has *stratified-witness* if it has no cycles through \exists -edges.⁵ 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 2.1. *The chase of any query Q with any set of DEDs with stratified-witness terminates (see Appendix A for the definition of the chase). The size of the resulting query is exponential in the maximum arity of a relation and the size of the resulting query is in $O(|Q|^{a^{l+1}})$.*

2.7. XML integrity constraints (XICs)

We designed a class of XML constraints so as to preserve the fundamental correspondence between query containment and constraint satisfaction which holds in the relational case for (unions of) conjunctive queries and (disjunctive) embedded dependencies. XICs have the same general form as (4), but the relational atoms are replaced by predicates defined by XPath expressions, just like in the case of XBind queries.

⁴The chase flow graph is similar to the graph used to determine the existence of stratified normal forms for ILOG programs [20]. These invent object identities, just like the chase invents new variables.

⁵The notion of a set of dependencies with stratified-witness first arose in a conversation between the first author and Lucian Popa. It was then independently used in [13] and in [15] (in the latter paper, under the term *weakly acyclic*).

Proposition 2.2. (a) For every disjunction-free 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 all XBind queries Q_1, Q_2 , there is a disjunction-free XIC $\text{cont}(Q_1, Q_2)$ such that for every instance I , $Q_1(I) \subseteq Q_2(I) \Leftrightarrow I \models \text{cont}(Q_1, Q_2)$.

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 $Q_2(\mathbf{x}) \leftarrow B_2(\mathbf{x}, \mathbf{z})$, $\text{cont}(Q_1, Q_2) = \forall \mathbf{x} \forall \mathbf{y} [B_1(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z} B_2(\mathbf{x}, \mathbf{z})]$.

2.8. Bounded AT-XICs

This is a class of XICs in whose presence the containment of AT-XBind queries is decidable (see Theorem 4.3 below). Of course, an AT-XIC is an XIC whose atom path expressions belong to AT-XPath. The boundedness condition is perhaps surprising, but we show in Theorem 6.2 below that this class is maximal, in the sense that containment becomes undecidable if we allow even modest use of unboundedness. Intuitively, boundedness means that existential quantification is disallowed over variables binding to attribute and text values, and is allowed only over nodes whose depth in the XML tree is bounded by the size of the XIC. Specifically, we say that an XIC variable v has bounded depth if it appears in some atom $[p](v)$, or $[p](v, w)$ or $[p](w, v)$ where: (i) p is an XPath expression consisting only of a chain of child navigation steps, and (ii) w has bounded depth. An XIC is bounded if all existentially quantified variables have bounded depth (there is no restriction on the universally quantified variables).

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 1, all XICs (2), (1), (3) are bounded. However, the following variation of XIC (2) is not, because variable a does not have bounded depth: $\forall p [//person](p) \rightarrow \exists a [./address](p, a)$.

3. Translation to a relational framework

Our strategy is to translate XML queries and constraints to relational queries and constraints. To this end we define a **generic relational encoding for XML** whose schema we call GReX. More specifically, we shall represent XML documents as certain relational instances⁶ over the schema

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

The “intended meaning” of the relations in GReX reflects the fact that XML data is a tagged tree. The unary predicate `root` denotes the root node of the XML document, and the unary relation `el` is the set of all nodes. `child` and `desc` are subsets of `el` \times `el` such that their second component is a child, respectively a descendant (including itself) of the first component.

⁶ We emphasize that this does not mean that the XML data is necessarily stored according to the relational schema GReX. Regardless of its physical storage, we reason about XML data using GReX as its virtual relational view.

Note that, for brevity of notation, our `desc` models XPath's descendant-or-self navigation axis. $\text{tag} \subseteq \text{el} \times \text{string}$ associates the tag in the second component to the node in the first. $\text{attr} \subseteq \text{el} \times \text{string} \times \text{string}$ gives the node, attribute name and attribute value in its first, second, respectively third component. $\text{id} \subseteq \text{string} \times \text{el}$ 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). $\text{text} \subseteq \text{el} \times \text{string}$ associates a node to the text inside it.

3.1. Relational translation of XPath expressions

We begin by moving outward (clearly preserving equivalence) the disjunction (`|` in paths, or in qualifiers), thus obtaining a *disjunction* of (`|`, or)-free XPath expressions. E.g., `/son|daughter` translates to `/son \cup /daughter`. Next, we translate these disjunction-free expressions into *conjunctions* of GReX atoms or equality atoms.

This is done using the operators $\mathcal{T}(c, p, n)$ and $\mathcal{Q}(c, q)$ defined in Fig. 2. Here c is the context node (which is ignored for `/p` and `//p`), p is a (`|`, or)-free path expression, n is a variable denoting a node in the node set yielded by p , and q is an or-free qualifier expression. (z, u below denote fresh variables, and s is a string constant.) Note that the translation of both value-based ($=$) and identity-based ($==$) equality conditions is the same.⁷

When the GReX is interpreted with the *intended meaning* described above, this translation corresponds exactly to the formal semantics in [34].

3.2. Relational translation of XBind queries and XICs

We saw how to translate XPath expressions to disjunctions of conjunctions of GReX or equality atoms. This immediately gives a translation of XBind queries into unions of conjunctive queries. For instance, the conjunctive queries corresponding to Xb_0 and Xb_1 in Example 1.1 are

$$B_0(a) \leftarrow \text{root}(r), \text{desc}(r, d), \text{child}(d, c), \text{tag}(c, \text{"author"}), \text{text}(c, a), \quad (5)$$

$$\begin{aligned} B_1(a, b, a1, t) \leftarrow & B_0(a), \text{root}(r), \text{desc}(r, d), \text{child}(d, b), \quad (6) \\ & \text{tag}(b, \text{"book"}), \text{child}(b, au), \text{tag}(au, \text{"author"}), \\ & \text{text}(au, a1), \text{child}(b, t), \text{tag}(t, \text{"title"}), a1 = a. \end{aligned}$$

Using the same translation of XPath expressions, we obtain a translation of XICs to DEDs. For instance, the constraint (2) in Example 1.2 translates to

$$\begin{aligned} \forall r, d, p \ (\text{root}(r) \wedge \text{child}(r, p) \wedge \text{tag}(p, \text{"person"})) \\ \rightarrow \exists a \ \text{child}(p, a) \wedge \text{tag}(a, \text{"address"}). \end{aligned}$$

⁷ For simplicity of presentation we assume that value-based equality $p_1 = p_2$ (and non-equality) is used only when p_1, p_2 end in attribute or text navigation steps, i.e. we do not check value-based equality of element nodes. We can easily allow this type of equality by extending GReX with a new relation `valueq`, and dependencies stating that `valueq` is an equivalence relation. All results presented in the paper hold for this case.

$$\begin{aligned}
\mathcal{T}(x, /p, y) &= \{\text{root}(u)\} \cup \mathcal{T}(u, p, y) \\
\mathcal{T}(x, //p, y) &= \{\text{root}(u), \text{desc}(u, z)\} \cup \mathcal{T}(z, p, y) \\
\mathcal{T}(x, p_1/p_2, y) &= \mathcal{T}(x, p_1, z) \cup \mathcal{T}(z, p_2, y) \\
\mathcal{T}(x, p_1//p_2, y) &= \mathcal{T}(x, p_1, z) \cup \{\text{desc}(z, u)\} \cup \mathcal{T}(u, p_2, y) \\
\mathcal{T}(x, p[q], y) &= \mathcal{T}(x, p, y) \cup \mathcal{Q}(y, q) \\
\mathcal{T}(x, ., y) &= \{x = y\} \\
\mathcal{T}(x, n, y) &= \{\text{child}(x, y), \text{tag}(y, "n")\} \\
\mathcal{T}(x, @n, y) &= \{\text{attr}(x, "n", y)\} \\
\mathcal{T}(x, @*, y) &= \{\text{attr}(x, z, y)\} \\
\mathcal{T}(x, \text{id}(p), y) &= \mathcal{T}(x, p, z) \cup \{\text{id}(z, y)\} \\
\mathcal{T}(x, \text{text}(), y) &= \{\text{text}(x, y)\} \\
\mathcal{T}(x, \text{id}(s), y) &= \{\text{id}(s, y)\} \\
\mathcal{T}(x, \text{ancestor-or-self}, y) &= \{\text{desc}(y, x)\} \\
\mathcal{Q}(x, q_1 \text{ and } q_2) &= \mathcal{Q}(x, q_1) \cup \mathcal{Q}(x, q_2) \\
\mathcal{Q}(x, p) &= \mathcal{T}(x, p, z) \\
\mathcal{Q}(x, p_1 == p_2) &= \mathcal{T}(x, p_1, z) \cup \mathcal{T}(x, p_2, z) \\
\mathcal{Q}(x, p_1 \neq p_2) &= \mathcal{T}(x, p_1, z) \cup \mathcal{T}(x, p_2, u) \cup \{u \neq z\} \\
\mathcal{Q}(x, p_1 = p_2) &= \mathcal{T}(x, p_1, z) \cup \mathcal{T}(x, p_2, z) \\
\mathcal{Q}(x, p_1 \neq p_2) &= \mathcal{T}(x, p_1, z) \cup \mathcal{T}(x, p_2, u) \cup \{u \neq z\} \\
\mathcal{Q}(x, p = s) &= \mathcal{T}(x, p, s)
\end{aligned}$$

Fig. 2. Relational compilation of AT-XPath expressions.

For the query reformulation problem we will also need to translate the views, which are expressed in XQuery. Translating the output of XQueries is more complicated and in fact we shall see that XQuery views are best translated into relational *constraints*—DEDs. We dedicate a large part of Section 5 to that task.

3.3. GReX models

The translations we just gave are semantically sound only when the GReX predicates are interpreted according to the *intended meaning* described above. Our goal however is to transfer reasoning about XBind queries and XICs into relational reasoning, preferably with DEDs, for which we can use the *chase* technique. However, it is not possible to capture the class of GReX-models with the intended meaning using just first-order logic. Indeed, neither the fact that `child` is the edge relation of a tree nor the fact that `desc` is the reflexive, transitive closure of `child` are first-order definable [14] (much less DED-definable!). On the other hand, the translations of AT-XBind queries and bounded XICs are quite special kinds of first-order formulas; they are not even as general as the class of all DEDs over GReX. This leaves room for pursuing our strategy.

3.4. TIX

We have identified a certain set of DEDs over the signature GReX that, although incapable of completely capturing the intended meaning, suffice to give us a chase-based

decision procedure for containment of *AT-XBind* queries, even under bounded XICs, as well as logical implication of bounded XICs. We call this set of dependencies **TI_X** (from **True In XML**):⁸

- (base) $\forall x, y [\text{child}(x, y) \rightarrow \text{desc}(x, y)],$
- (trans) $\forall x, y, z [\text{desc}(x, y) \wedge \text{desc}(y, z) \rightarrow \text{desc}(x, z)],$
- (refl) $\forall x [\text{el}(x) \rightarrow \text{desc}(x, x)],$
- (el_c) $\forall x, y [\text{child}(x, y) \rightarrow \text{el}(x) \wedge \text{el}(y)],$
- (el_d) $\forall x, y [\text{desc}(x, y) \rightarrow \text{el}(x) \wedge \text{el}(y)],$
- (el_{id}) $\forall s, x [\text{id}(s, x) \rightarrow \text{el}(x)],$
- (el_r) $\forall x [\text{root}(x) \rightarrow \text{el}(x)],$
- (someTag) $\forall x [\text{el}(x) \rightarrow \exists t \text{tag}(x, t)],$
- (oneTag) $\forall x, t_1, t_2 [\text{tag}(x, t_1) \wedge \text{tag}(x, t_2) \rightarrow t_1 = t_2],$
- (keyId) $\forall s, e_1, e_2 [\text{id}(s, e_1) \wedge \text{id}(s, e_2) \rightarrow e_1 = e_2],$
- (oneAttr) $\forall x, n, v_1, v_2 [\text{attr}(x, n, v_1) \wedge \text{attr}(x, n, v_2) \rightarrow v_1 = v_2],$
- (noLoop) $\forall x, y [\text{desc}(x, y) \wedge \text{desc}(y, x) \rightarrow x = y],$
- (oneParent) $\forall x, y, z [\text{child}(x, z) \wedge \text{child}(y, z) \rightarrow x = y],$
- (oneRoot) $\forall x, y [\text{root}(x) \wedge \text{root}(y) \rightarrow x = y],$
- (topRoot) $\forall x, y [\text{desc}(x, y) \wedge \text{root}(y) \rightarrow \text{root}(x)],$
- (inLine) $\forall x, y, u [\text{desc}(x, u) \wedge \text{desc}(y, u) \rightarrow x = y \vee \text{desc}(x, y) \vee \text{desc}(y, x)],$
- (choice) $\forall x, y, z [\text{child}(x, y) \wedge \text{desc}(x, z) \wedge \text{desc}(z, y) \rightarrow x = z \vee y = z].$

4. Deciding containment of *AT-XBind* queries

Our first important result justifies the development given in Section 3.

Theorem 4.1. *Let B_1 and B_2 be two *AT-XBind* queries and let the **GR_EX**-conjunctive queries $c(B_1)$ and $c(B_2)$ be their translations. Then, B_1 is contained in B_2 over all XML documents if and only if $c(B_1)$ is contained in $c(B_2)$ over all **GR_EX**-instances that satisfy **TI_X**.*

The proof is in Appendix B. This result then yields decision procedures for containment of *XBind* queries from our fragments.

Theorem 4.2.

1. *Deciding containment for *AT-XBind* queries is Π_2^P -complete,*
2. *containment for *LAT-XBind* queries is NP-complete and*

⁸ A collection D_1, \dots, D_n of XML documents is represented by the disjoint union of schemas X_i and the union of constraints in each **TI_X_{*i*}**, where each X_i (**TI_X_{*i*}**) is obtained from X (resp. **TI_X**) by subscripting all relational symbols with i .

3. the *LAT-XBind* fragment is maximal, in the following sense: extending it with any subset of the additional features allowed in the *AT-XBind* fragment yields Π_2^p -hardness of containment.

The upper bounds in parts (1) and (2) follow from Theorem 4.1 by chasing $c(B_1)$ with *TI*X (the chase is guaranteed to terminate by Proposition 2.1). According to Theorem A.2 in Appendix A, the containment test for part (1) reduces to checking the existence of exponentially many containment mappings, whence the Π_2^p upper bound. The *LAT-XBind* fragment is easier because its translation does not involve union, and the chase with (*inLine*) and (*choice*) does not apply, so the containment test reduces to finding only one containment mapping (whence the NP upper bound). The lower bounds in Theorem 4.2 are inherited from the lower bounds established in [9] for containment for fragments of XPath, which in turn come from similar lower bounds for (unions of) conjunctive queries [5,29]. Hence, part (3) is not surprising when considering disjunction or non-equalities. A bit more surprising are the observations that *ancestor-or-self* can be translated away by introducing disjunction, and that descendant navigation together with equality on node identities can be used to simulate *ancestor-or-self*.

Proposition 4.1. *Given any XBind query B and set D of bounded AT-XICs, let $c(B)$ and $c(D)$ be the corresponding relational translations. Then the chase of $c(B)$ with $c(D)$ will either*

1. produce the atoms $\text{child}(x, y)$, $\text{desc}(y, x)$ for some variables x, y , or
2. an equality atom $c_1 = c_2$ with c_1, c_2 distinct constants, or
3. terminate.

Note that if either of (1) or (2) come to hold, B is unsatisfiable (returns the empty answer over all documents satisfying D).

Theorem 4.3. *Containment of AT-XBind queries under bounded AT-XICs is decidable. If we fix the set of XICs, containment is in Π_2^p .*

The proof of this theorem follows easily from Theorem A.2 in Appendix A, as well as the following generalization of Theorem 4.1 to containment under bounded AT-XICs:

Theorem 4.4. *Let B_1 and B_2 be two AT-XBind queries and let the GReX-conjunctive queries $c(B_1)$ and $c(B_2)$ be their translations. Let X be a set of bounded AT-XICs and $c(X)$ the set of DEDs obtained from its translation. Then, B_1 is contained in B_2 over all XML documents satisfying X if and only if $c(B_1)$ is contained in $c(B_2)$ over all GReX-instances that satisfy $\text{TI}X \cup c(X)$.*

See Appendix B for the proof of Theorem 4.4. With this reduction we can then chase with *TI*X and (in view of Proposition 4.1) the relational translation of the XICs.

5. AT-XBind query reformulation with AT-XQuery views

The strategy of our algorithm is to translate each XBind query to a union of conjunctive queries, translate all XQuery views (not just their XBind parts; the output part as well!) to

DEDs and apply the C&B algorithm (which we introduced in [8] and extended in [13]), to the resulting relational problem of reformulation of unions of conjunctive queries under DEDs.

5.1. Relational query reformulation: the C&B algorithm

5.1.1. 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 unions of conjunctive query views can be captured by DEDs 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:

$$(c_V) \quad \forall x \forall y \forall z [A(x, y) \wedge B(y, z) \rightarrow V(x, z)],$$

$$(b_V) \quad \forall x \forall z [V(x, z) \rightarrow \exists y A(x, y) \wedge B(y, z)],$$

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

5.1.2. Review of C&B

Assume that in addition, the following dependency holds on the database (it is an inclusion dependency): $(ind) \quad \forall x \forall y [A(x, y) \rightarrow \exists z B(y, z)]$. Suppose that we want to reformulate the query $Q(x) \leftarrow A(x, y)$.

First, Q is *chased* with all available dependencies, until no more chase steps apply (see Appendix A for a detailed definition of the chase). The resulting query is called the *universal plan*. In our example, a chase step with (ind) yields $Q_1(x) \leftarrow A(x, y), B(y, z)$, which in turn chases with (c_V) to the universal plan $Q_2(x) \leftarrow A(x, y), B(y, z), V(x, z)$. 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) .

5.2. Translating schema correspondences

5.2.1. Obstacles in capturing XQuery views with dependencies

In [8,13] we point out that conjunctive query views can be captured using two inclusion dependencies and hence (minimal) rewriting with views becomes minimization under dependencies. We cannot capture XQuery views with two inclusion dependencies in the same way because they are more expressive: (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 1.1 (see [7] for more details).

Nested, correlated subqueries: Recall that the navigation part of an XQuery is described by a set of decorrelated XBind queries. Also recall that every XBind query can be straightforwardly translated to a union of conjunctive queries over schema GReX. This can now be captured with inclusion dependencies (in fact with two DEDs) as in [8,13]. For XBind queries Xb_0, Xb_1 in Example 1.1, we obtain B_0, B_1 defined by queries (5), respectively (6). Here is one of the four resulting dependencies:

$$\begin{aligned} \forall a [B_0(a) \rightarrow \exists r \exists d \exists c \text{root}_1(r) \\ \wedge \text{desc}_1(r, d) \wedge \text{child}_1(d, c) \wedge \text{tag}_1(c, \text{"author"}) \wedge \text{text}_1(c, a)]. \end{aligned}$$

Construction of new elements: *Element nodes with constant identity.* Recall Q from Example 1.1, which constructs a *unique result* element node, which is a child of the root and whose identity does not exist anywhere in the input document, but rather is an invented value. The invented identity does not depend on the bindings of Q 's variables, i.e. it is a constant. We shall represent this constant as a function of no arguments F_{result} described by the unary relation G_{result} whose intended meaning is given by $G_{\text{result}}(x) \Leftrightarrow x = F_{\text{result}}()$. This meaning is captured by the following dependencies (note that we model the XML documents corresponding to the output and input of the XQuery view as relational instances of schema GReX₂, respectively GReX₁):

$$\exists y G_{\text{result}}(y), \tag{7}$$

$$\forall y_1 \forall y_2 [G_{\text{result}}(y_1) \wedge G_{\text{result}}(y_2) \rightarrow y_1 = y_2], \tag{8}$$

$$\forall r \forall c [\text{root}_2(r) \wedge \text{child}_2(r, c) \wedge \text{tag}_2(c, \text{"result"}) \rightarrow G_{\text{result}}(c)], \tag{9}$$

$$\forall c [G_{\text{result}}(c) \rightarrow \exists r \text{root}_2(r) \wedge \text{child}_2(r, c) \wedge \text{tag}_2(c, \text{"result"})]. \tag{10}$$

5.2.2. *Element nodes whose identity depends on the variable bindings*

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} .⁹ We capture this function by extending the schema with the relational symbol G_{item} , intended as the graph of F_{item} ($G_{\text{item}}(x, y) \Leftrightarrow y = F_{\text{item}}(x)$) and use dependencies to enforce this intended meaning.

$$\forall x_1 \forall x_2 \forall y [G_{\text{item}}(x_1, y) \wedge G_{\text{item}}(x_2, y) \rightarrow x_1 = x_2], \tag{11}$$

$$\forall x \forall y_1 \forall y_2 [G_{\text{item}}(x, y_1) \wedge G_{\text{item}}(x, y_2) \rightarrow y_1 = y_2], \tag{12}$$

$$\forall x [B_0(x) \rightarrow \exists y G_{\text{item}}(x, y)], \tag{13}$$

⁹ Many semistructured and XML query languages use functions like F_{item} as explicit query primitives, under the name of Skolem functions. Our technique for translating into DEDs fits seamlessly with an extension of XQuery with Skolem functions.

$$\forall x \forall c [G_{\text{item}}(x, c) \rightarrow \exists r G_{\text{result}}(r) \wedge \text{child}_2(r, c) \wedge \text{tag}_2(c, "item")], \quad (14)$$

$$\forall a \forall w [G_{\text{writer}}(a, w) \rightarrow \text{text}_2(w, a)], \quad (15)$$

$$\forall a \forall w [G_{\text{writer}}(a, w) \wedge \text{desc}_2(w, d) \rightarrow d = w]. \quad (16)$$

F_{item} is an injective function by (11) and (12). The domain of F_{item} contains the set of bindings for $\$a$ (13). The range of F_{item} consists of `item` nodes that are children of the `result` node (14). The contents of the `writer` elements is the text $\$a$ was bound to (15), and the `writer` node has no children (16).

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_t^a 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 $\{F_t^a\}_{a,t}$ by the relation $C: \forall a \forall t F_t^a(n, n') \Leftrightarrow C(a, t, n, n')$. Again, we capture the intended meaning for C using DEDs. We illustrate only one of these DEDs. DED (17) 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 GRex_2):

$$\forall a \forall t \forall n \forall n' \forall d [C(a, t, n, n') \wedge \text{desc}_1(n, d) \rightarrow \exists d' \text{desc}_2(n', d') \wedge C(a, t, d, d')]. \quad (17)$$

5.3. The algorithm

5.3.1. Plans: reformulations using auxiliary schema

If any variables of the XBind query Xb are bound to element nodes, then Xb cannot be reformulated against the storage schema S because the node identities in the storage and published data are disjoint. This is because the semantics of the XQuery views specifies that for each input XML document, a new XML document is created. 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. We have shown in Section 5.2 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 (for Example 1.1, Aux includes G_{item}). Then any plan can be represented by a query against the extended storage schema $S \cup \text{Aux}$. However, we have to be careful when doing so: note that, since the relations in Aux model functions, they must be treated as relations with limited binding patterns [23] in which the invented node identity is an output, all other attributes are inputs. We say that a plan is *viable* if all variables appearing on input positions of Aux relations also occur in relations from S .

Algorithm for XBind reformulation**Given:**

- an AT-XBind query Xb ,
- a schema correspondence described by a set of AT-XQuery views \mathbf{V} (in both directions),¹⁰
- the set \mathbf{C}_X of bounded AT-XICs over the various XML documents,
- the set \mathbf{C}_R of integrity constraints over the relational part of the schema;

Do:

- translate Xb to the union of conjunctive queries $c(Xb)$;
- translate 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 as in Section 5.2);
- translate \mathbf{C}_X to the set $c(\mathbf{C}_X)$ of DEDs;
- compute the set \mathbf{R} of reformulations *against* $S \cup \text{Aux}$ by applying the C&B algorithm to $c(Xb)$ under $\text{TIX} \cup c(\mathbf{V}) \cup c(\mathbf{C}_X) \cup \mathbf{C}_R$.

Return:

- all *minimal* (see below) queries in \mathbf{R} that correspond to a *viable* (see above) reformulation plan.

End.**5.3.2. Minimality**

We first define it for conjunctive queries. Let C be a set of relational constraints (e.g., DEDs). We say that a conjunctive 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 . Clearly, if a query is not already C -minimal then there exists a C -minimal query that is equivalent to it under C . Correspondingly, we define a union of conjunctive queries to be minimal if (i) none of the conjunctive queries in the union is contained in another, and (ii) by removing any relational atom from any conjunctive query in the union (even if we add arbitrarily many equalities instead), we compromise the equivalence to the original union. To define minimal XBind queries, substitute in the above “relational atom” with “individual navigation step within the XPath expression p appearing in some atom $[p](x, y)$ ”.

Theorem 5.1 (*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 iff $c(R)$ is a minimal reformulation of $c(Xb)$ under $\text{TIX} \cup c(\mathbf{V}) \cup c(\mathbf{C}_X) \cup \mathbf{C}_R$.*

The proof is in Appendix C.

¹⁰ As in all data integration scenarios where the schema correspondence is given by exact views, we assume that distinct LAV views (used for adding redundancy subsequently, during tuning) have disjoint target schemas. We assume also that the integrity constraints do not relate these schema portions. These assumptions hold by default in publishing, where the integrity constraints are expressed in terms of the original proprietary schema (published by a GAV view), and the LAV views are added subsequently, during tuning, to model redundant data.

We combine this result with the following result given in [13] which states the completeness of the C&B algorithm:¹¹

Theorem 5.2 (Deutsch and Tannen [13]). *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 . Let R be any query that is D -minimal and is equivalent to Q under D . Then, R is isomorphic to a subquery of U .*

Remark. In fact, we need an extension of this result that holds when Q is a union of conjunctive queries and D is a set of *disjunctive* embedded dependencies (DEDs). The extension is straightforward, details are given in [7].

Theorem 5.1 and (the extension of) Theorem 5.2 imply.

Corollary 5.1 (overall completeness). *The algorithm finds all minimal reformulations for AT-XBind queries, under AT-XQuery views, bounded AT-XICs and stratified-witness relational dependencies.*

6. Calibrating the results

We investigate what happens if we attempt to relax the restrictions we have put on XPath, XBind, XQuery and XICs.

6.1. Justifying the restrictions in the containment results

The result we find most intriguing is about the effect of adding wildcard child navigation to the fragment whose containment problem is in NP.

Theorem 6.1. *Extending the LAT-XBind fragment with wildcard child navigation (*) raises the complexity of containment from NP- to Π_2^P -complete.*

The proof is in Appendix D. It follows that the containment of such queries cannot be decided by simply chasing with TIX. In fact, unless $NP = \Pi_2^P$, it follows that no addition of disjunction-free embedded dependencies to TIX can give us a theorem similar to Theorem 4.1. We conjecture that adding DEDs won't help either. To obtain the upper bound, we had to devise a more complex algorithm for deciding containment of these queries (see Appendix D).¹²

The next result says that bounded AT-XICs are the maximal class of constraints for which AT-XBind containment is decidable. Its proof is in Appendix E.

¹¹ The journal version of this result will be presented elsewhere.

¹² Interestingly, the wildcard algorithm works also for parent and ancestor navigation [9].

Theorem 6.2. *Containment of AT-XBind queries under unbounded AT-XICs is undecidable.*

Remark. XICs cannot express *all* the constraints captured by DTDs. In fact, DTDs and bounded AT-XICs do not mix well: a modification of the previous result can be used to show that containment of AT-XBind queries under bounded AT-XICs and DTDs is also undecidable [7,9]. (Actually, this is how we prove Theorem 6.2, see Appendix E.) In [9] we also have the result that containment of AT-XBind queries under DTDs is PSPACE-hard. Since then, this was improved (even with wildcard navigation) to EXPTIME-completeness, using tree automata techniques [26].

6.2. Justifying the restrictions in the reformulation algorithm

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* as the following reduction shows:

Proposition 6.1. *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 C .*

The proof of Proposition 6.1 is in Appendix F. Notice that 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 in general 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 [22].

Proposition 6.1 allows us to transfer the lower bounds on the problem of deciding query containment for various XBind classes to the problem of finding minimal reformulations of XBind queries. It follows therefore from Theorem 6.1 that even modest use of non-AT-XBind features such as wildcard child navigation makes any NP algorithm for finding minimal reformulations incomplete unless $NP = \Pi_2^P$. A careful analysis of our C&B-based algorithm shows that it can find a minimal reformulation in NP for queries hence these considerations apply to it. It also follows from Theorem 6.2 that there is no complete

algorithm for finding minimal reformulations of *AT-XBind* queries in the presence of even modest use of XICs from outside the bounded class.

7. Related work

The results presented here have already been announced in [9,13]. The completeness of our reformulation algorithm relies on the completeness of the C&B algorithm [8] that was given in [13] and its proof will be detailed elsewhere (currently, it can be found in [7]).

Clearly, XPath containment is a particular case of (single-atom) XBind containment. XPath containment for a fragment corresponding to the *LAT-XPath* fragment without any equality conditions, but extended with wildcard child navigation was shown to be coNP-complete by Miklau and Suciu [25]. The upper bound proof technique is similar to our proof of Theorem 6.1. In both cases, containment is characterized by the existence of exponentially many containment mappings. The difference is that, for the fragment of Miklau and Suciu, each mapping is found in PTIME (as the expressions are acyclic), while for Theorem 6.1, finding the containment mappings is NP-complete (the queries are cyclic due to use of variables). This explains the jump from coNP to Π_2^P . Wood [35] shows the decidability of deciding containment of various XPath fragments under DTDs and a special class of integrity constraints. Neven and Schwentick [26] have solved the problem of containment of XPath expressions with wildcard under DTDs, showing the problem to be EXPTIME-complete (using automata-theoretic techniques). They also consider XPath expressions with variables, showing that their containment (in the absence of DTDs) is PSPACE-complete. The apparent discrepancy between this result and our Π_2^P -completeness result for *AT-XBind** queries (which have variables too) is due to the fact that [26] considers variables to be free in the XPath expression, and bound in an outside context. Containment is decided under all possible contexts (hence universal semantics), which means treating variables as constants. In contrast, we defined in [9] an existential semantics that allows us to bind a variable to a node reached by XPath navigation, and reuse that binding in another part of the XPath expression. This is the natural semantics needed to generalize from containment of XPath expressions to containment of XBind queries, and it was devised keeping in mind the larger goal of XBind reformulation.

8. 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 (*AT-XBind* queries) and integrity constraints (bounded *AT-XICs* and stratified-witness DEDs). The algorithm can be reused for reformulation of XICs. 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.

8.1. Practicality of the approach

There are of course XQuery features we cannot translate 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 here holds for any query that is compilable relationally. Features beyond *AT-XPath* that are compilable relationally are navigation along the *parent*, *ancestor*, *previous-sibling*, *following-sibling*, *previous* and *following* axes. While reasoning completely about document order is a challenging research issue, we show in [9] a partial result saying that all of our decision algorithms extend if we allow *previous-sibling* and *following-sibling* navigation and add appropriate axioms to *TIX*.

We have built a query reformulation system [7] based on the method presented here. Putting these ideas to work required a good deal of challenging engineering but, as reported in [12], 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. We thank the reviewer of this submission for the very insightful comments.

Appendix A. Chasing with DEDs

Let φ be a conjunction of relational, equality and non-equality atoms whose terms are variables or constants. We call φ a \neq -conjunction. We denote with $vars(\varphi)$ ($const(\varphi)$) the set of variables (constants) appearing in φ . Given a set of variables $\bar{u} \subseteq vars(\varphi)$ and denoting $\bar{v} = vars(\varphi) \setminus \bar{u}$, $\varphi(\bar{u})$ denotes the query defined by the formula $\exists \bar{v} \varphi$. We will interchangeably refer to φ as the FO formula represented by the conjunction of its atoms, or as the set of atoms per se. We will therefore use the notation $a \in \varphi$ to say that the atom a appears in φ . We say that \neq -conjunction φ is *consistent* iff (i) its equality atoms do not imply¹³ the equality $c_1 = c_2$, where $c_1, c_2 \in const(\varphi)$, and (ii) for each non-equality atom $x \neq y$, φ 's equality atoms do not imply $x = y$.

Given \neq -conjunctions φ_1, φ_2 , a \neq -homomorphism from φ_1 to φ_2 is a mapping h from $vars(\varphi_1) \cup const(\varphi_1)$ to $vars(\varphi_2) \cup const(\varphi_2)$ such that

- $h(c) = c$ for each $c \in const(\varphi_1)$,
- for each relational atom $R(\bar{w}_1) \in \varphi_1$, where \bar{w}_1 is a tuple of variables and constants, there exists an atom $R(\bar{w}_2) \in \varphi_2$ such that the component-wise equality $h(\bar{w}_1) = \bar{w}_2$ is implied by the equality atoms in φ_2 (via reflexivity, symmetry and transitivity).

¹³ Via reflexivity, symmetry and transitivity.

- for each equality atom $x = y \in \varphi_1$, with x, y variables or constants, the equality $h(x) = h(y)$ is implied by the equality atoms in φ_2 ,
- for each non-equality atom $x \neq y \in \varphi_1$, with x, y variables or constants, there exists a non-equality atom $u \neq v \in \varphi_2$ such that $h(x) = u$ and $h(y) = v$ are implied by the equality atoms in φ_2 .

$$\text{Let } d \text{ be the DED} \quad \forall \bar{x} \psi \rightarrow \bigvee_{i=1}^l \exists \bar{y}_i \xi_i, \quad (18)$$

where ψ is a \neq -conjunction with $\bar{x} = \text{vars}(\psi)$, and for each i , ξ_i is a \neq -conjunction with $\bar{y}_i \subseteq \text{vars}(\xi_i) \subseteq \bar{x} \cup \bar{y}_i$. Let φ be a \neq -conjunction and assume w.l.o.g. that $\text{vars}(\varphi) \cap \text{vars}(d) = \emptyset$ and that $\forall i \neq j \bar{y}_i \cap \bar{y}_j = \emptyset$ (this can always be achieved by renaming the variables in d). We say that a *chase step* of φ with d *applies* iff there is a \neq -homomorphism h from ψ to φ such that for each i , h has no extension to a \neq -homomorphism from $\psi \wedge \xi_i$ to ψ . In other words, there is no \neq -homomorphism h' from $\psi \wedge \xi_i$ to ψ such that $h'(x) = h(x)$ for each $x \in \bar{x}$. The *result* of this chase step is a disjunction of \neq -conjunctions obtained as follows: First obtain the disjunction $\bigvee_{i=1}^l \varphi \wedge h'(\xi_i)$, where h' is a mapping on $\text{vars}(d)$ that extends h to be the identity on \bar{y}_i . Next, remove from this disjunction all inconsistent \neq -conjunctions. Note that all CQ \neq s may be inconsistent, so the result of the chase step is the unsatisfiable empty disjunction \perp .

Example A.1. Consider the DED

$$\forall x \forall y R(x, y) \rightarrow \exists z S(x, z) \wedge z \neq x \vee \exists u T(y, u) \quad (19)$$

Then no chase step with DED (19) applies to $R(m, n) \wedge T(n, o)$ because the only \neq -homomorphism $h = \{x \mapsto m, y \mapsto n\}$ from $R(x, y)$ to $R(m, n)$ has an extension to $R(x, y) \wedge T(y, u)$, namely $\{x \mapsto m, y \mapsto n, u \mapsto o\}$. However, a chase step applies to $R(m, n)$, yielding $R(m, n) \wedge S(m, z) \wedge z \neq m \vee R(m, n) \wedge T(n, u)$. Note that no inconsistent disjuncts were created in this case.

We lift the definition of chase step of a \neq -conjunction to that of a disjunction D of \neq -conjunctions. Let $D = \bigvee_j \varphi_j$, and let δ be a DED such that, again w.l.o.g., $\text{vars}(\delta) \cap \text{vars}(D) = \emptyset$. Then a chase step of D with δ applies iff there is a j_0 such that a chase step with δ applies to the \neq -conjunction φ_{j_0} , yielding the result $\text{step}_\delta(\varphi_{j_0})$. The result of the chase step on D is defined as $\bigvee_{j \neq j_0} \varphi_j \vee \text{step}_\delta(\varphi_{j_0})$.

Given a set of DEDs Δ and a disjunction of \neq -conjunctions D , we say that the chase of D with Δ *terminates* iff there exists a sequence of chase steps with DEDs from Δ which starts from D and yields a disjunction of \neq -conjunctions U for which no chase step applies anymore. We call U a result of chasing D with Δ , denoted $\text{chase}_\Delta(D)$. Note that the chase with DEDs is not confluent, so there may be several terminating chase sequences, with distinct results. $\text{chase}_\Delta(D)$ refers to a non-deterministically picked result.

Theorem A.1. *Let D be a set of DEDs and d a single DED of general form (18). Assume that the chase of ψ with D terminates, yielding the following disjunction of conjunctions with equality and non-equality atoms $\text{chase}_D(\psi) = \bigvee_{j=1}^m \psi_j$. Then $D \models d$ iff for each $j \in \{1, \dots, m\}$ there is an $i \in \{1, \dots, l\}$ and a \neq -homomorphism from ξ_i to ψ_j .*

Theorem A.2. Let D be a set of DEDs, Q_1 a union of conjunctive queries with non-equalities (UCQ $^\neq$), and Q_2 a UCQ $^\neq$ query $Q_2(\bar{y}) \leftarrow \bigvee_l \varphi_l(\bar{y})$. Assume that the chase of Q_1 with D terminates, yielding the UCQ $^\neq$ query $U(\bar{x}) \leftarrow \bigvee_{k=1}^m \psi_k(\bar{x})$. Then Q_1 is contained in Q_2 under D iff for each k , there exists an l and a \neq -homomorphism h from φ_l to ψ_k , such that $h(\bar{y}) = \bar{x}$.

Appendix B. Proof sketch for Theorems 4.1 and 4.4

Theorem 4.1. We say that an instance of the relational GReX is **intended** if it obeys the intended meaning described in Section 3. Of course, there is a one-to-one correspondence between XML documents and such intended instances. Intended instances evidently satisfy TIX, but moreover they exhibit two crucial features: (i) the `child` relation corresponds to a tree and (ii) `desc` is the reflexive, transitive closure of `child`. Clearly, the containment of B_1, B_2 holds on all XML documents if and only if it holds for $c(B_1), c(B_2)$ on all intended GReX instances. Since the latter satisfy the constraints in TIX, the “if” direction of the theorem follows trivially. Observe however that there are unintended instances which nevertheless satisfy TIX (e.g. ones in which `desc` contains pairs of nodes not connected by a chain of `child` edges).

“only if”:¹⁴ Assuming B_1 is contained in B_2 (i.e. $c(B_1)$ is contained in $c(B_2)$ over intended GReX instances) we show that $c(B_1)$ is contained in $c(B_2)$ over all TIX instances. By the classical chase theorem [1], it suffices to show that there is a containment mapping from $c(B_2)$ into the result U of chasing $c(B_1)$ with TIX. The chase is guaranteed to terminate by Proposition 2.1. \square

The canonical instance is not intended. Denoting with $CI(U)$ the canonical instance of U (the database instance obtained from U by treating all variables as constants [1]) observe that $CI(U)$ is a GReX instance. Moreover, since no more chase step with constraints from TIX applies to U , $CI(U)$ is a TIX instance as well. Note that, if $CI(U)$ were an intended instance as well, then by our assumption, $c(B_2)$ would necessarily have a mapping into $CI(U)$ which would correspond to the sought containment mapping. However, $CI(U)$ is not necessarily an intended instance: feature (ii) above may be violated by descendant navigation steps from B_1 which correspond in $c(B_1)$ (and hence in U) to atoms `desc`(x, y) such that there is no path of `child` edges from x to y . Let’s call such atoms *unsupported*.

The supported instance $SI(U)$. Let $SI(U)$ be the instance obtained as a copy of $CI(U)$ to which we add atoms as follows: for every unsupported atom `desc`(x, y) in $CI(U)$ add `child`(x, y) to $SI(U)$.

Claim $SI(U)$ is intended. *Proof of claim.* We first show that (ii) is satisfied. Since no chase step of $CI(U)$ applies with (base), (refl), (trans) and all the (el_X) axioms, it follows

¹⁴For simplicity, we only sketch here the proof for the case when B_1 is satisfiable (has a non-empty answer on some XML document). We also show only the case when $c(B_1), c(B_2)$ and the result of chasing $c(B_1)$ with TIX are conjunctive queries (free of disjunction and inequalities). See [7] for the extension of the chase to unions of conjunctive queries and DEDs, and for the proof of Theorem 4.1 in the general case.

that `desc` contains the reflexive, transitive closure of the `child` relation. It is easy to check that by construction, all `desc` atoms in $SI(U)$ are supported, so `desc` is precisely the closure of `child`. We next show that $SI(U)$ satisfies `TIX`, i.e. no chase step with a `DED` from `TIX` was enabled by the addition of `child` atoms to $CI(U)$. For this observe that the only chase steps that may be triggered this way are with (*oneParent*), (*choice*) and (*elC*). A case analysis shows that if any such chase step applied in $SI(U)$, using the fact that $CI(U)$ satisfies `TIX` we would derive the contradiction that `desc`(x, y) must be supported in $CI(U)$. (i) follows from the satisfiability of B_1 and the fact that (*noLoop*) holds in $SI(U)$. A few minor points to show are that each element node in $SI(U)$ has precisely one tag, unique attribute names, etc., all of which are guaranteed because no more chase step with `TIX` applies. *End of proof of claim.* \square

The claim and our assumption imply the existence of a containment mapping m from $c(B_2)$ into $SI(U)$. But B_2 is an *AT-XBind* query, so all `child` navigation steps test for a specific tag name. (Here it is crucial that wildcard `child` navigation is absent.) Therefore, all atoms `child`(x, y) in $c(B_2)$ are accompanied by some atom `tag`($y, "t"$), with t a constant. But observe that for any unsupported atom `desc`(x, y), the chase with (*someTag*) adds an atom `tag`(y, f), where f is a fresh variable, not mentioned anywhere else in U . We conclude that the `child` atoms from $SI(U) \setminus CI(U)$ cannot serve as image of atoms from $c(B_2)$, so m is really a containment mapping from $c(B_2)$ into $CI(U)$.

Generalization to Theorem 4.4. The proof is essentially the same as for Theorem 4.1, with two modifications: (1) modify the claim to state that $SI(U)$ is an intended instance satisfying the set of *XICs* X , and (2) observe that the `tag`(y, f) atoms added during the chase are still guaranteed to contain fresh variables f . This is because X contains only *AT-XICs*, which can never match against the `tag`(y, f) atoms, and therefore cannot introduce equalities between f and any other variable or constant in U .

Appendix C. Proof sketch of Theorem 5.1

Theorem 4.1 (proven in Appendix B) can be (easily) generalized to containment in the presence of bounded *AT-XICs*, *DEDs* stemming from compiling *AT-XQuery* views, and arbitrary *DEDs* over the proprietary relational schema. The important additional observation is that the supported instance $SI(U)$ (notation refers to proof in Appendix B) satisfy all constraints because no chase step applies, as all *DEDs* either stem from *AT-XPath* expressions or do not mention *GRex* at all. From this generalization, it follows immediately that, if the chase terminates, R is a reformulation of Xb if and only if $c(R)$ is a reformulation of $c(Xb)$.

The termination of the chase follows from the observation that all *LAV* views (used for adding redundancy subsequently, during tuning) have distinct target schemas, disjoint from the original proprietary schema, and therefore not mentioned in any of the integrity constraints pertaining to the proprietary schema. It follows that $c(V) \cup C_R \cup \text{TIX}$ have stratified witness so the chase with only these *DEDS* terminates by Proposition 2.1. It also follows that if adding $c(C_X)$ results in a divergent chase, then the chase with only $c(C_X) \cup \text{TIX}$ must diverge as well. But this is excluded by Proposition 4.1. \square

Appendix D. Proof sketch for Theorem 6.1

Let's denote *AT-XPath* expressions extended with wildcard child navigation with *AT-XPath** and the resulting *XBind* queries with *AT-XBind**. For the compilation of *AT-XBind** queries just add the rule $\mathcal{T}(x, *, y) = \{\text{child}(x, y)\}$.

It turns out that the DEDs in TIX become insufficient in reasoning about wildcard navigation. A counterexample to Theorem 4.1 is given by $/ * // *$ and $// * / *$: these are obviously equivalent, yet their relational compilations are not equivalent under TIX. The problem could be fixed by adding to TIX the axiom $\forall x, y, z \text{ child}(x, y) \wedge \text{desc}(y, z) \rightarrow \exists u \text{ desc}(x, u) \wedge \text{child}(u, y)$. However, no extension of TIX can cover the following counterexample which will play a central role in the lower bound proof.

Example D.1. There are *AT-XPath** expressions p, p' such that (the boolean *XBind* query) $Xb() := [p](x)$ is contained in $Xb'() := [p'](x)$ but the relational compilation $c(Xb)$ of Xb is not contained in $c(Xb')$ under TIX:

$$p = / [b/1[@x = "1"]] \\ [a[a[@x = "1"]][*/\text{descendant-or-self} :: a][c][*/ * [c][*/ *][x = "0"]] \\ [b/0[@x = "0"]]$$

$$p' = / [. // * [a[a][c]][*/ * [c][*/ *][@x = /b/ * /@x]]$$

A graphical representation of p and p' appears in Fig. 3, in which we depict child navigation steps with single arrows and *descendant-or-self* navigation steps with double, dashed arrows. The tag names are used to label the nodes (* is used for wildcards), and solid non-arrow lines associate attributes with nodes. $@x = 0$ indicates that the string value of the x -attributes is “0”. The dotted line represents an equality condition on x -attributes. To see that Xb is contained in Xb' , observe that $a/ */ \text{descendant-or-self} :: a$ in p is equivalent to $(a/a)|(a/ */ /a)$, and hence Xb is equivalent to $Q_1() := [p_1](x) \cup Q_2 := [p_2](x)$ where p_1, p_2 are obtained by replacing the subpath $a/ */ \text{descendant-or-self} :: a$ with a/a , respectively $a/ */ /a$ in p . But both Q_1, Q_2 are contained in Xb' , as witnessed by two containment mappings from $c(Xb')$: one matching the x -attributes in $c(Xb')$ against the “0”-valued x -attributes in $\mathcal{T}(p_1)$ and one matching them against the “1”-valued x -attributes of $\mathcal{T}(p_2)$. On the other hand, according to the chase theorem [1], $c(Xb)$ is not contained in $c(Xb')$ under TIX because there is no containment mapping from $c(Xb)$ into the result of chasing $c(Xb)$ with TIX. \square

Upper bound. First, observe that $// = \bigcup_{0 \leq k} *^k$, where $*^k$ is short for the concatenation of k wildcard navigation steps. More generally, every *AT-XBind** query p with n occurrences of $//$ is equivalent to an infinite union of $//$ -free queries: denoting with $p(k_1, \dots, k_n)$ the result of replacing the i th occurrence of $//$ in p with $*^{k_i}$, p is equivalent to $\bigcup_{0 \leq k_1, \dots, 0 \leq k_n} p(k_1, \dots, k_n)$. Therefore, the containment of p in p' reduces to checking the containment of each $p(k_1, \dots, k_n)$ in p_2 . The key results making our containment decision procedure possible are that (i) each individual containment can be decided according to Theorem 4.1, (Proposition D.1) and (ii) it is sufficient to check the containment for only finitely many $//$ -free *XBind* queries in the union (Proposition D.2).

- We also add n copies of a *universal gadget* (one copy for every x_i). The universal gadget (defined shortly) is denoted $U(x)$ and it is a $AT\text{-XPath}^*$ expression having occurrences of $@x$ for some attribute name x . For every x_i , the corresponding copy of U has $@x$ substituted with $@x_i$, denoted $U(x_i)$.

This completes the construction of q_1 , up to the specification of the universal gadget. First we show the construction of $p_2() := q_2$, where q_2 is the query body. q_2 contains $l + m + n$ subexpressions:

- for every $1 \leq i \leq l$, q_2 contains $[/C_i](c_i)$, $[@u_i](c_i, u_i)$, $[@v_i](c_i, v_i)$, $[@w_i](c_i, w_i)$ where, as before, u_i, v_i, w_i are the variables occurring in clause C_i .
- for every $1 \leq j \leq m$, q_2 contains the atom $[/y_j/@y_j](y_j)$.
- for every $1 \leq i \leq n$, q_2 contains a *satisfaction gadget* $[S(x_i)](x_i)$. $S(x)$ denotes $AT\text{-XPath}^*$ expression with occurrences of $@x$ for some x (defined shortly).

We now specify the universal and satisfaction gadgets. Recalling counterexample D.1, $U(x)$ is a copy of p , and $S(x)$ is a copy of p' modified to return the attribute $@x$: $[// * [a[a][c]][*/* [c][*/*][@x = /b/*/*@x]/@x]]$.

We still have to prove that this construction is a reduction. According to Proposition D.2, $p \subseteq p'$ if and only if $p(k_1, \dots, k_n) \subseteq p'$ both for $k_i = 0$ and $k_i > 0$. Recalling the discussion in counterexample D.1, the containment mapping corresponding to $k_i = 0$ binds x_i to “0”, while that corresponding to $k_i > 0$ binds x_i to “1”. Moreover, it is easy to see that any containment mapping from p' to p corresponds to a satisfying assignment of ϕ . Therefore, $p_1 \subseteq p_2$ if and only if every truth assignment to the x_i s has an extension to the y_j s that satisfies all clauses of ϕ (or, equivalently, if and only if ϕ is valid). \square

Appendix E. Proof sketch for Theorem 6.2

By reduction from the following undecidable problem: Given context-free grammar $G = (\Sigma, N, S, P)$ where Σ is the set of terminals (containing at least two symbols), N the non-terminals, $S \in N$ the start symbol, $P \subseteq N \times (\Sigma \cup N)^*$ the productions, and $L(G)$ the language generated by G , the question whether $L(G) = \Sigma^*$ is undecidable [19].

In fact, it is simpler to present a reduction to containment in the presence of bounded $AT\text{-XICs}$ and DTDs and we do so below. However, a careful analysis of the used DTD features reveals that these are captured as $XICs$ of two forms: $\forall x [//A](x) \rightarrow \exists y [./A](x, y) \vee \exists y [./B](x, y)$ and $\forall x [//A](x) \rightarrow \exists y [./@s](x, y)$. These are not bounded $AT\text{-XICs}$: note the illegal existential quantification of y .

The reduction. Given context-free grammar $G = (\Sigma, N, S, P)$, we construct an instance $(DTD_G, D_G, XP_1 \subseteq XP_2)$ such that XP_1 is contained in XP_2 over all XML documents conforming to the description DTD_G and satisfying the $XICs$ in D_G if and only if $\Sigma^* \subseteq L(G)$. We first show DTD_G , which does not exercise all features of DTDs. The features of DTD_G used to prove undecidability can be easily shown to be fully captured by $XICs$:

```
<!ELEMENT B (A|E)>
<!ELEMENT A (A|E)>
<!ELEMENT E (PCDATA)>
<!ATTLIST B i #ID, S #IDREFS>
<!ATTLIST A i #ID, sym (a1|a2|...|an), N1 #IDREFS, ..., Nk #IDREFS>.
```

B, E, A are fresh names, a_1, \dots, a_n are the alphabet symbols in Σ , N_1, \dots, N_k are the nonterminals in N . Every document conforming to DTD_G is a list (unary tree) of elements, whose head is tagged B and unique leaf tagged E . The inner elements (if any) of the list are tagged A , and their *sym* attribute contains a symbol of Σ . Every document conforming to DTD_G thus corresponds to a word $w \in \Sigma^*$, and every pair s, t of A -elements such that t is a descendant of s determines a substring of w .

The set of XICs D_G (shown shortly) is designed such that, whenever a document conforms to the DTD_G and satisfies D_G , the following *claim* holds: for every pair s, t of A -elements with t a descendant of s , let u be the corresponding substring of w (if $s = t$, u is the unit length string given by the value of t 's *sym* attribute). Then for every $1 \leq j \leq k$ such that there is a derivation of u starting from nonterminal N_j , the value of the attribute i of t is a token of the value of the N_j attribute of s .¹⁵ Furthermore, the S attribute of the B -element contains all tokens of the S attribute of the first A -element, if any.

We omit the proof of the claim, but illustrate for the grammar $S \rightarrow cS \mid cc$ and word $w = ccc$. An XML document corresponding to w which conforms to DTD_G and satisfies the claim is

```
<B i="0" S="2 3">
<A sym="c" i="1" S="2 3"><A sym="c" i="2" S="3">
<A sym="c" i="3" S=""> <E>any text goes here</E></A></A></A></B>.
```

Now we have $w \in L(G)$ if and only if there is a derivation of w in G starting from S , which by the claim is equivalent to the i -attribute in the parent of the E -element being among the tokens of the S -attribute in the B -element. Therefore, $\Sigma^* \subseteq L(G)$ is equivalent to the containment $[/[/E]/@i \subseteq /B/@S$ which we pick for $XP_1 \subseteq XP_2$.

We now show the XICs D_G . For every production $p \in P$, we construct an XIC ($prod_p$) as illustrated by the following example. Let R, T be nonterminals and a, b alphabet symbols in the production $R \rightarrow aRbT$. The corresponding XIC is

$$(prod_p) \quad \forall x, y \quad [./A[@sym = "a"]/id(@R)/A[@sym = "b"]/id(@T)/@i](x, y) \rightarrow [./@R](x, y).$$

We enforce that the tokens in the S -attribute of the first A -element be included in the S -attribute of the B -element with the XIC

$$(start_B) \quad \forall x, y \quad [/B](x) \wedge [./A/@S](x, y) \rightarrow [./@S](x, y).$$

Furthermore, we may assume without loss of generality that G has at most one ε -production, namely $S \rightarrow \varepsilon$ (see the procedure for elimination of ε -productions employed when bringing a grammar in Chomsky Normal Form [19]). If $S \rightarrow \varepsilon \in P$, add to D_G the XIC

$$(d_\varepsilon) \quad \forall x, y \quad [/B](x) \wedge [./@i](x, y) \rightarrow [./@S](x, y). \quad \square$$

¹⁵ Recall that an IDREFS attribute a models a *set* of IDREF attributes, represented by the set of whitespace-delimited tokens of a 's string value.

Appendix F. Proof of Proposition 6.1

Let C be any class of relational instances. We reduce the problem CON to the problem MIN where

CON: given conjunctive queries Q_1, Q_2 , decide whether Q_1 is contained in Q_2 on a class of models C such that Q_2 returns a non-empty answer for at least one model in C (denoted $Q_1 \subseteq_C Q_2$)

MIN: given conjunctive query Q and set of embedded dependencies D , decide if Q is minimal under all models from C that satisfy D .

Our definition of conjunctive queries allows equality and constants so queries may be unsatisfiable. The condition that Q_2 return a non-empty answer on at least some model $I \in C$ is easy to check in all common scenarios. When C is the class of all instances, the canonical instance of Q_2 is an example for I . When C is specified by a set of stratified-witness dependencies, the result of chasing Q_2 with these dependencies is an example, as long as it does not equate two constants, in which case Q_2 is equivalent to the empty query.

First we reduce CON to the auxiliary problem DISJ where,

DISJ: given conjunctive queries P_1, P_2 , decide whether $P_1 \subseteq_C P_2$ or $P_2 \subseteq_C P_1$.

Let $Q_1(\mathbf{x}) \leftarrow \text{body}_1(\mathbf{x}, \mathbf{y})$ and $Q_2(\mathbf{x}) \leftarrow \text{body}_2(\mathbf{x}, \mathbf{z})$ be conjunctive queries over schema S . Here \mathbf{x} denotes a tuple of variables x_1, \dots, x_n , and similarly for \mathbf{y}, \mathbf{z} . Let e be a fresh, nullary predicate. We extend instances by interpreting e in two ways: one as the empty set, and the other as the singleton empty tuple. We denote with C^e the class of models obtained by extending every model in C in both ways. Define $Q_1^e(\mathbf{x}) \leftarrow \text{body}_1(\mathbf{x}, \mathbf{y}), e()$. We claim that

$$Q_1 \subseteq_C Q_2 \Leftrightarrow Q_1^e \subseteq_{C^e} Q_2 \vee Q_2 \subseteq_{C^e} Q_1^e.$$

Proof of Claim. Let $I \in C$, and let J be the extension of I with an interpretation for e . Notice first that since Q_2 is not defined in terms of e , $Q_2(J) = Q_2(I)$ regardless of e 's interpretation. Moreover, $Q_1^e(J) = Q_1(I)$ when e is interpreted as non-empty, and $Q_1^e(J) = \emptyset$ otherwise.

\Rightarrow : Pick an arbitrary $J \in C^e$ and let $I \in C$ be J 's restriction. If e is the empty set, $Q_1^e(J) = \emptyset \subseteq Q_2(J)$. When e is interpreted as the singleton empty tuple, $Q_1^e(J) = Q_1(I) \subseteq Q_2(I) = Q_2(J)$. \Leftarrow : Q_2 returns a non-empty answer on at least one instance $I \in C$. Then $Q_2 \subseteq_{C^e} Q_1^e$ is false. Indeed, Q_2 must have a non-empty answer also on the extension J of I with the empty set e . But $Q_1^e(J) = \emptyset$, which contradicts the containment statement.

It must therefore be the case that $Q_1^e(J) \subseteq Q_2(J)$ is true for all $J \in C^e$, in particular for those in which e is interpreted as non-empty, but on these, $Q_1(I) = Q_1^e(J) \subseteq Q_2(J) = Q_2(I)$. Since the corresponding set of restrictions I of J is exactly C , we have $Q_1 \subseteq_C Q_2$. (End of proof of claim.) \square

Now we reduce DISJ to MIN. Denote $P_1(\mathbf{x}) \leftarrow \text{body}_1(\mathbf{x}, \mathbf{y})$ and $P_2(\mathbf{x}) \leftarrow \text{body}_2(\mathbf{x}, \mathbf{z})$. Let D be the set of dependencies $\{c_1, b_1, c_2, b_2\}$ over the schema in which we added P_1, P_2

as relational symbols:

(c_i) $\forall \mathbf{x} \forall \mathbf{y} [body_i(\mathbf{x}, \mathbf{y}) \rightarrow P_i(\mathbf{x})]$,

(b_i) $\forall \mathbf{x} [P_i(\mathbf{x}) \rightarrow \exists \mathbf{y} body_i(\mathbf{x}, \mathbf{y})]$.

Notice that, on any instance satisfying D , the relation P_i contains exactly the result of the query P_i . Also notice that c_i, b_i are exactly the kind of dependencies we use in the C&B approach to capture views. We claim that

$$P_1 \subseteq_C P_2 \vee P_2 \subseteq_C P_1 \\ \Leftrightarrow \\ P(\mathbf{x}) \leftarrow P_1(\mathbf{x}), P_2(\mathbf{x}) \text{ is not minimal over } C\text{-instances satisfying } D.$$

Notice that, on any instance satisfying D , P defines the intersection of P_1 and P_2 when regarded as queries.

Proof of Claim. Since P has only two atoms in its body, it is not minimal if and only if it is equivalent to either $M_1(\mathbf{x}) \leftarrow P_1(\mathbf{x})$ or $M_2(\mathbf{x}) \leftarrow P_2(\mathbf{x})$. But this is true if and only if when regarded as queries, $P_1 =_C P_1 \cap P_2$ or $P_2 =_C P_1 \cap P_2$, if and only if $P_1 \subseteq_C P_2$ or $P_2 \subseteq_C P_1$. \square

References

- [1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Reading, 1995.
- [2] C. Beeri, M.Y. Vardi, A proof procedure for data dependencies, J. ACM 31 (4) (1984) 718–741.
- [3] P. Buneman, S. Davidson, W. Fan, C. Hara, W.-C. Tan, Keys for XML, in: WWW10, 2001.
- [4] M. Carey, J. Kiernan, J. Shanmugasundaram, E. Shekita, S. Subramanian, XPERANTO: middleware for publishing object-relational data as XML documents, in: VLDB, 2000.
- [5] A. Chandra, P. Merlin, Optimal implementation of conjunctive queries in relational data bases, in: STOC, 1977.
- [6] S.S. Cosmadakis, P.C. Kanellakis, Functional and inclusion dependencies, in: Advances in Computing Research, Vol. 3, 1986, pp. 163–184.
- [7] A. Deutsch, XML query reformulation over mixed and redundant storage, Ph.D. Thesis, University of Pennsylvania, CIS Department, 2002.
- [8] A. Deutsch, L. Popa, V. Tannen, Physical data independence, constraints and optimization with universal plans, in: VLDB, 1999.
- [9] A. Deutsch, V. Tannen, Containment and integrity constraints for XPath fragments, in: KRDB 2001, September 2001.
- [10] A. Deutsch, V. Tannen, Containment for classes of XPath expressions under integrity constraints, Technical Report MS-CIS-01-21, University of Pennsylvania, 2001.
- [11] A. Deutsch, V. Tannen, Optimization properties for classes of conjunctive regular path queries, in: DBPL, 2001.
- [12] A. Deutsch, V. Tannen, Mars: a system for publishing XML from mixed and redundant storage, in: VLDB, 2003.
- [13] A. Deutsch, V. Tannen, Reformulation of XML queries and constraints, in: ICDT, 2003.
- [14] H.D. Ebbinghaus, J. Flum, Finite Model Theory, Springer, Berlin, 1995.
- [15] R. Fagin, P. Kolaitis, R. Miller, L. Popa, Data exchange: semantics and query answering, in: ICDT, 2003.
- [16] M. Fernandez, A. Morishima, D. Suciu, Efficient evaluation of XML middle-ware queries, in: SIGMOD 2001, May 2001.

- [17] M. Fernandez, WangChiew Tan, D. Suciu, SilkRoute: trading between relations and XML, in: WWW9 Conference, May 2000.
- [18] A. Halevy, Answering queries using views: a survey, *VLDB J.* 10 (4) (2001) 270–294.
- [19] J. Hopcroft, J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, 1979.
- [20] R. Hull, M. Yoshikawa, ILOG: declarative creation and manipulation of object identifiers, in: *VLDB*, 1990.
- [21] M. Lenzerini, Data integration: a theoretical perspective, in: *PODS*, 2002.
- [22] A. Levy, A.O. Mendelzon, Y. Sagiv, D. Srivastava, Answering queries using views, in: *Proceedings of PODS*, 1995.
- [23] A. Levy, A. Rajaraman, J. Ullman, Answering queries using limited external query processors, in: *Proceedings of PODS*, 1996.
- [24] I. Manolescu, D. Florescu, D. Kossman, Answering XML queries on heterogeneous data sources, in: *Proc. of VLDB 2001*, 2001.
- [25] G. Miklau, D. Suciu, Containment and equivalence for an XPath fragment, in: *Proceedings of PODS*, 2002.
- [26] F. Neven, T. Schwentick, XPath containment in the presence of disjunction, DTDs and variables, in: *ICDT*, 2003.
- [27] C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, 1994.
- [28] R. Pottinger, A.Y. Halevy, Minicon: a scalable algorithm for answering queries using views, *VLDB J.* 10 (2–3) (2001) 182–198.
- [29] Y. Sagiv, M. Yannakakis, Equivalences among relational expressions with the union and difference operators, *J. ACM* 27 (1980) 633–655.
- [30] J. Shanmugasundaram, J. Kiernan, E.J. Shekita, C. Fan, J. Funderburk, Querying XML views of relational data, in: *VLDB*, September 2001.
- [31] W3C, XML Path Language (XPath) 1.0. W3C Recommendation 16 November 1999. Available from <http://www.w3.org/TR/xpath>.
- [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>.
- [34] P. Wadler, A formal semantics of patterns in XSLT, in: *Proceeding of the Conference for Markup Technologies*, 1999.
- [35] P.T. Wood, Containment for XPath fragments under DTD constraints, in: *ICDT*, 2003.