HAL
archives-ouvertes.fr

Types and constraints: from relational to XML data

Nicole Bidoit

▶ To cite this version:

Nicole Bidoit. Types and constraints: from relational to XML data. Semantics in Data and Knowledge Bases - 4th International Workshops, SDKB 2010, Revised Selected Papers., Jun 2010, Bordeaux, France. 2011. <hal-00641909>

HAL Id: hal-00641909
https://hal.inria.fr/hal-00641909

Submitted on 17 Nov 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Types and constraints: from relational to XML data

Nicole L. Bidoit-Tollu

Univ. Paris-Sud - UMR CNRS 8623 - INRIA Saclay
LRI Database Group & INRIA Leo Team
nicole.bidoit@lri.fr

Abstract. The goal of this article is to show that, in the context of XML data processing, information conveyed by schema or XML types is a powerful component to deploy optimization methods. We focus on the one hand on recent work developed for optimizing query and update evaluation for main-memory engines and on the other hand on techniques for checking XML query-update independence. These methods are all based on static type analysis. The aim of the article is to show how types rank before constraints for XML data processing and the presentation of each method is kept informal.

Keywords: XML, Schema, Type, Query, Update, Optimization.

1 Introduction

Integrity constraints have been considered from the beginning as a key component of information systems and databases. They are fundamental not to say mandatory for specifying the semantic of data. Integrity constraints provide a way to fill the gap between information and data. Integrity constraints also raise many problems and beginning with efficient constraint checking and enforcing integrity constraints, repairing inconsistent database states, etc. They play a major role in the process of query optimization, called semantic query optimization, where the constrains known to hold on the database are used to rewrite queries into equivalent and more efficient ones. Integrity constraints and rewriting techniques based on constraints have also been successful for developing query rewriting using views, for data exchange, peer data exchange, data integration, etc.

Types can also be viewed as constraints, but in a different fashion. Intuitively, in the database world, integrity constraints are content-based properties whereas types are structural properties. A type consists of a possibly infinite set of values and also, in programming languages, of a set of operations on those values. A type may be atomic or structured. Roughly, a composite/structured type is build using existing types and type constructors like record, set, list, etc. In the context of databases, types have been essentially investigated in the context of the object-oriented database model where typing [28,29] shares features of conventional object-oriented programming languages and of taxonomy systems where data is organized in hierarchy of classes and subclasses. Typing helps to detect programming errors which is of major importance for developing database applications and managing huge amount of data.

There is a very large body of research on using constraints for query optimization. However, contrary to programming languages, typing in database has been rarely investigated for optimization purpose. The aim of the article is to give some evidence that, in the context of XML data processing, types are becoming as important as constraints[1], and provide ways to optimize XML queries and updates as well as it provides ways to develop static analysis for checking other critical properties such as query update independence.

The article is organized as follows. The next section is an introductory discussion on constraints and types and covers the relational model and XML data. The rest of the article is dedicated to present some ongoing work investigating optimization techniques for XML document processing based on static analysis, making an extensive use of the information given by types or schemas. The presentation is devoted to pieces of work that have been conducted in the Leo team [4]. It certainly does not aim at covering all issues and existing studies. The style of the presentation remains informal and mainly relies on examples. Section 3 addresses query optimization and Section 4 update optimization for main-memory engines. Section 5 focuses on update query independence.

2 Preliminaries

Let us start by revisiting the relational model. Types in the relational model are tremendously simple. Given a set of attributes U, a set of atomic values, called domain, is assigned to each attribute. A relational database schema over U is given by a set of relation names $\mathscr{R}$ and a mapping (a type assignment) τ assigning to each relation name r a finite subset of U[2]. Thus, a relational type is a finite set of attributes. For instance, the relational schema defined by $\mathscr{R}$=$\{R, S\}$ with $\tau(R)$=$\{A, B, C\}$ and $\tau(S)$=$\{A, D\}$ is usually denoted $\mathscr{R}$=$\{R(A, B, C), S(A, D)\}$. Relational types are of course useful for writing queries either using the algebra or SQL as well as for writing updates. They are also useful for checking well-typedness of queries and infer their output type. For instance, the expression $\pi_{AC}(\sigma_{D='foo'}([R] \bowtie [S]))$ is well typed wrt to the schema $\mathscr{R}$ above. The output type for this expression is $\{A, C\}$. For instance, [37] investigates how the principal type of a relational expression can be computed, giving all possible assignments of types to relation names under which the expression is well-typed. This study is extended to the nested relational calculus in [36]. However, relational types are quite "poor". They are equivalent to record types in programming languages. More sophisticated types are associated to the non first normal form model and to the complex object model model [11]. Integrity constraints have been introduced in order to increase the semantic contents of relational databases. Amon integrity constraints, the following are the most studied classes: functional dependencies [34,51], equality generating dependencies and tuple generating dependencies [15]. Integrity constraints, as opposed to relational types, can be used for query optimization. The chase [13,45] is probably

[1] The frontier between types and constraints may is also not as clear-cut as for relational databases

[2] For now, we leave integrity constraints out of the specification of a database schema.

one of the most known example of a technique based on dependencies and allowing for minimizing the number of joins of an algebraic query expression, and recent investigation has proved its efficiency in practice [48]. Integrity constraints, especially functional dependencies, are also very useful for optimizing updates: schema normalization [33], avoiding update anomalies, can be viewed as some kind of update optimization.

In the relational world, types are almost totally useless for optimization purpose. Constraints have a better place whereas physical data structures and access methods provide the real solutions to optimization problems in practice [50].

Semi-structured data [25,12] and XML data [2] are classically introduced as self-describing and schema-less: in its general setting, semi-structured data are abstracted by labelled graphs; well-formed XML documents are ordered labelled trees as described for instance by the *Document Object Model* [41]. Simple tree representations of XML documents are given in Fig. 1. Interestingly enough, almost every published article whose scope is XML management with a database perspective, makes the assumption of a schema or type constraining the structure of XML documents when they do not investigate schemas and types for XML themselves. XML schemas aim at defining the set of allowable labels and also the way they can be structured. In the first place, XML schemas are meant to distinguish meaningful documents from those that are not and provides the user with a concrete semantic of the documents. The document on the left hand side of Fig. 1 is well-formed: it is a labelled tree. However, this document is not valid wrt the DTD described by the following rules: $doc \rightarrow a*$, $a \rightarrow bc*d$, $b \rightarrow String$, $d \rightarrow e$. These rules specify that the root of the document should be labelled by doc, that the children of the root should be labelled by a (it is not the case for the document considered here because the root node has its last child labelled by e) and the children of a nodes should be first a node labelled by b followed by possibly several nodes labelled c, and finally a node labelled d, etc. Figure proposes another example of a DTD and a document valid wrt that DTD.

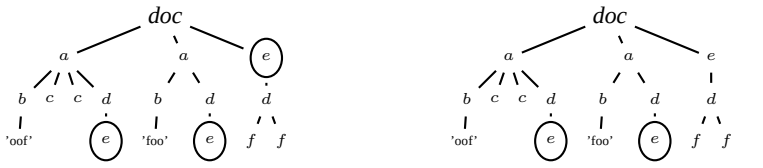
XML schemas turn out to bring many advantages for validation, data integration, translation, etc They are very useful for the specification of meaningful queries over XML data and can be used for query optimization. The main practical languages for describing XML types are *Document Type Definition (DTD)* which is indeed part of the original definition of XML [2] and is probably the most widely used, and *XML Schema* [52,24] which is a more elaborate, not to say overly complex, schema notation based on XML itself. Many other proposals for schema languages have flourished: DSD [43], RelaxNG [32], Shematron [42], Assertion Grammars [49]. See for instance [47,44] for a formal classification and a comparison of these languages.

What about XML integrity constraints? It is often claimed that schemas and constraints are both properties that restrict the allowed documents and thus that the frontier between schemas or types and constraints is not as clear-cut as for relational databases. Although ID attributes in a DTD can be used to uniquely identify an element within an XML document, these attributes can not be used to express constraints such as keys. Although XML schemas allow to specify keys or references in terms of Xpath [18], such style of specification is rather intricate especially when it comes to reasoning about constraints. Indeed, constraints for semi-structured data and XML [38,39], keys and foreign

keys for XML [26,27] have received a lot of attention from the database community. Update anomalies and normal forms for XML documents have been investigated in [14]. Unified formalisms for capturing schema and integrity constraints for XML have also been studied in order to reason about types and constraints [19].

As opposed to relational databases, optimization techniques based on integrity constraints have not yet been investigated whereas schema or type-based optimization techniques are already used [46] in query engines.

To conclude this preliminary section, we briefly proceed to a short discussion on *element type*, a notion that will be used in the next sections. Each optimization techniques discussed next uses the information provided by a schema, namely a DTD, in order to assign types to the elements of a document valid wrt to the schema. Considering a document t and an element or node n in t, the type of n is, in its simplest form, its label. An alternative to this very simple notion of element type is to consider the chain of labels that match the path from the document root node of t to n. The former notion is that used in Sections 3 and 4, resp. for query and update optimization. The latter is central to Section 5 for dealing with query-update independence. Obviously, the notion of element type based on labels is less precise than the notion of element type based on chains although this last one is still not fully precise. Let us illustrate this with the example of Fig. 1: on the left side, the circled nodes of the document are typed by the label e; on the right side, the circled nodes of the (same) document are typed by the chain $doc.a.d.e$; the chain type is more precise as it allows to distinguish e nodes that can be reached by navigating along $doc.a.d.e$ paths from those that cannot be reached that way; however, it is not precise enough to make the distinction between e nodes whose parent has at least one sibling labelled by c from those whose parent has no such sibling. Thus, in some sense, both notions of element type, label and chain, are approximations.



nodes typed by the label e nodes typed by the chain $doc.a.d.e$

Fig. 1. Labels and chains as type approximation for XML element.

3 Type-based optimization for XML queries

XQuery [9] is now well-recognized as a flexible and powerful language allowing to query collections of XML documents. As such, it is used not only for querying XML databases but also for instance to process XML files. In the latter context, main-memory

XQuery engines like Galax [3], Saxon [8], QizX [6,5], and eXist [1] are used to query files as they avoid building secondary storage structures. The price to pay or more exactly the problem to face is main-memory limitations of these systems (see [46]) because, in such configuration, the whole document is required to be loaded in main-memory before query processing. Main-memory XQuery engines are unable to process very large documents. Projecting XML document is an optimization technique introduced by [46] to address such limitation. The simplicity of the method is one of the key of its efficiency. Given a query Q over an XML document t, the idea is to determine, in a static manner, that is by analyzing the query expression and, if available, the type or schema of the queried documents, which part t' of the document t is needed in order to evaluate Q, and then, at loading time, to prune the document t; finally the query Q is evaluated in main-memory against the pruned document t'. This technique is quite powerful because in general, queries are very selective and only a small part of the initial document is relevant to the evaluation of the query. The core of the method, which determines its efficiency, resides in the static analysis of the query Q: how precise is it?

All the examples developed next are build using the DTD D and the XML document t depicted in Figure 2.

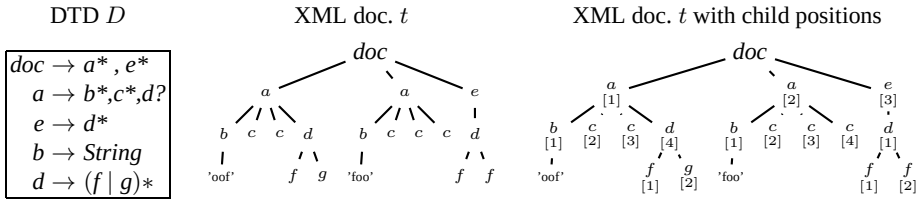


Fig. 2. Running example : a DTD D and a document t

The initial proposal of [46] determines the data needs of the query Q by extracting all path expressions in Q. These paths are evaluated at loading time to prune the initial document. This technique does not require any information about the structure of the document. This advantage raises some pruning overhead and imprecision especially when // axis occurs in query paths. For instance, consider the simple XQuery *for* $\$x$ *in /doc//[c] return* $<\!/yes>$ and the document t of Fig. 2. The path extracted is */doc//node()* because filters are not analyzed, resulting in projecting out the whole document t. Assuming that the filter $[c]$ of the query is taken into account by the analysis, the extracted path would be given by */doc//c*. This time, the projected document will be precise (and indeed is the result of the query Q) however, at loading time, the whole document will be explored by the query path, thus causing some pruning execution overhead. It is clear that, in this case, the knowledge of the DTD D for the document t, gives the opportunity to refine the extracted path */doc//c* to */doc/a/c* which avoids during projection for visiting and checking e elements for instance. This is what typed-based projection is about [17]. The type-based query scenario is as follows:

1. from the query Q and DTD D, a type projector π is inferred,
2. the document t, valid wrt D, is projected following π in a streaming manner, at loading time,
3. and finally, the query Q is evaluated over $\pi(t)$.

The main result [17] is that $Q(t)$=$Q(\pi(t))$.

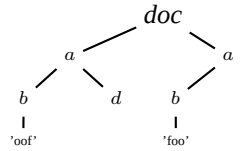
The type projector is specified by a set of node types given by labels. Execution of a type projector at loading time is straightforward. It consists of projecting nodes whose labels (types) belong to the type projector. Extracting a type projector π for a query Q given a DTD D is a two phase process whose first step extracts the paths of the query Q and the second step strongly relies on the type information given by D to generate the expected set of node types.

Example 1. For the query Q specified below, the paths extracted are */doc/a*, */doc/a//d* and */doc/a//text()*. The type extraction based on the DTD of Fig. 2 allows one to generate the type projector π=$\{doc, a, b, d,$ *String*$\}$ because nodes of type d that are descendants of nodes of type a are indeed children of nodes of type a and nodes of type *String* that are descendants of nodes of type a are children of nodes of type b.

Query Q

```
for $x in /doc/a
where $x//d
return $x//text()
```

Projected document $\pi(t)$



One can notice here that the evaluation of the projector avoids exploring the right most subtree of the document (Fig. 2) whose root is labelled by e even though this subtree contains nodes of type d.

Experiments have been done showing the benefits of the type-based projection optimization for queries. The method provides significant improvement of performance both in terms of memory consumption and time.

4 Type-based optimization for XML updates

In this section, we show that, type-based projection can also be used to optimize main-memory XML update processing. The update language considered is XQuery Update Facility (XUF) [10] with its two phase semantics: given a complex update expression U and a document t, the first phase generates a so-called update pending list, a list of elementary updates, and the second phase evaluates the update pending list over the document t. We focus on main-memory XUF processing which of course suffers of the same limitation as main-memory XQuery evaluation: XUF main-memory engines fails to process very large documents. For instance, eXist [1], QizX/open [6] and Saxon

[7] are unable to update documents whose size is greater than 150 MB (no matter the update query at hand).

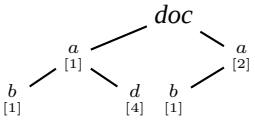
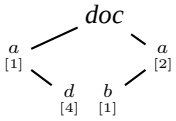
The scenario and type-based projection described for XML queries, cannot be applied directly for updates. Indeed, it is rather obvious to see that updating the projection of a document t will not produce the expected result unless the projection equals the whole document which is, of course, not very promising from an optimization point of view. We have developed in [20,21] a type-based optimization method for update. The update scenario is different from that for query optimization and is now composed of four steps:

1. from the update U and the DTD D, a type projector π is inferred,
2. the document t, valid wrt D, is projected following π in a streaming manner, at loading time,
3. the update U is evaluated over the projection $\pi(t)$ and produces a partial result $U(\pi(t))$,
4. finally, in order to produce the final result $U(t)$, the initial document t is merged with $U(\pi(t))$, in a streaming manner, at writing-serializing time.

The projector extracted for updates, a 3-level type projector, has been designed in order to capture update expression requirements and also in order to support the *Merge* phase. The 3-level projector is specified by 3 sets of types given by labels. The projection of the document t registers, in main-memory, the child position of nodes. No rewriting of the update U is necessary. The *Merge* step can be seen as a synchronized parsing of both the initial document t and the partial result $U(\pi(t))$. This requires a very small amount of memory. Only child positions of nodes and the projector π are checked in order to decide whether to output elements of t or of $U(\pi(t))$. No further changes are made on elements after the partial update: output elements are either elements of the original document t or elements of $U(\pi(t))$.
We start by explaining the last step of the update scenario (*Merge*) and then we outline the main ideas behind the 3-level type projector specification. The presentation remains informal, it does not cover all aspects of the method, even important ones, and relies on examples.

Example 2. Let us consider the simple update U whose expression is given in the left part of the next figure. This update intends to delete b children of a nodes having at least one d child. Here, the projector is meant to extract from the document the part which is used either to prepare the update or which is the target of the changes made by the update. For this example and for sake of simplicity, we use a (1-level) projector π=$\{doc, a, b, d\}$.
The *Merge* step parses in a synchronized manner both the input document t and the partial result $U(\pi(t))$. The rightmost part of Figure 2 shows the document t with child positions marked in square brackets under node labels. Wrt our example, executing *Merge* over t and $U(\pi(t))$ is quite obvious until after processing the node $a[1]$ of t together with the node $a[1]$ of $U(\pi(t))$. The two next nodes parsed by *Merge* are: the node $b[1]$, child of $a[1]$ in t and the node $d[4]$, child of $a[1]$ in $U(\pi(t))$. The child positions of these nodes and the information that the label a belongs to the type projector π allows one to identify that the node $b[1]$ should not be output: the initial document t

is further parsed without producing any element in the output. Then, the two next nodes examined by *Merge* are: the node $c[2]$, child of $a[1]$ in t and the node $d[4]$, child of $a[1]$ in $U(\pi(t))$. This time, the child positions of these nodes and the information that the label c does not belong to the type projector π are used to output the node $c[2]$ and further parse the initial document t.

Update U	Projected doc. $\pi(t)$	Partial update of $U(\pi(t))$
for $\$x$ in /doc/a where $x/d return delete $x/b		

The main issues that motivated the specification of the new type projector are: dealing with update expressions and ensuring correctness and efficiency of the *Merge* step. The type projector is specified by a tuple $\pi=(\pi_{\mathbf{no}}, \pi_{\mathbf{olb}}, \pi_{\mathbf{eb}})$ where each set π_x is a set of labels. Thus here, as for query optimization, element types are approximated by labels. The behavior of the 3-level projector is as follows:

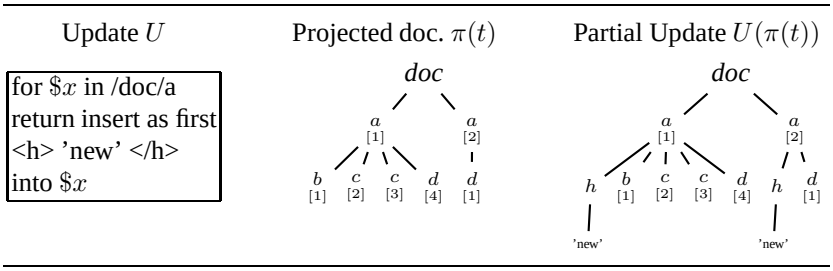
- A node of the document t whose label belongs to $\pi_{\mathbf{no}}$ is projected. The children of such nodes will be candidate for projection depending on their label and π.
- A node of the document t whose label belongs to $\pi_{\mathbf{olb}}$ is projected together with all its children even though their labels do not belong to π. This $\pi_{\mathbf{olb}}$ component of the projector is introduced in order to correctly handle insertion at *Merge* time as well as mixed-content. This is illustrated by the next example.
- A node of the document t whose label belongs to $\pi_{\mathbf{eb}}$ is projected together with all its descendants, even though their labels do not belong in π. Introducing this $\pi_{\mathbf{eb}}$ component of the projector is required for dealing with element extraction for instance in the context of moving some elements.

Indeed, the two components $\pi_{\mathbf{olb}}$ and $\pi_{\mathbf{eb}}$ bring more precision and efficiency. They minimize the projection size and speed up the type projector execution by avoiding to put in the projector the labels of some of the nodes that need to be projected. Interestingly enough, the 3-level projector designed for updates provides significant improvements for pure queries.
The extraction of the 3-level projector from the update is based on a sophisticated extraction of paths from the update expression which is followed by a type derivation relying on the DTD D.

Example 3. Let us consider the update U below which intends to insert an h element as first child of any node labelled by a. The 3-level projector extracted from this update is given by: $\pi_{\mathbf{no}}=\{doc\}$, $\pi_{\mathbf{olb}}=\{a\}$ and $\pi_{\mathbf{eb}}=\{\}$. Thus, while projecting the document t given in Fig. 2 wrt π all children of the nodes labelled a are projected. Note that neither the type c nor the type d occur in π. Note also that the right most subtree of

the tree whose root is labelled by e is not projected because e does not belong to the projector. When updating the projection $\pi(t)$, the new elements are inserted without child position. It should be understood that the positions in $\pi(t)$ correspond to child position of the nodes in the input document t.

Now, when merging the document t with the partial update $U(\pi(t))$, because all children of the node $a[1]$ have been projected, it is straightforward to output, in the final result, the children of $a[1]$ in the correct expected order.



Extensive experiments of the type-based optimization method for updates have been conducted and their results validate the effectiveness of the approach. They show that the technique succeeds in its primarily purpose: making possible to update very large documents with main-memory systems. Interestingly enough, the tests show that even when projection is not necessary because of memory limitation, using projection can reduce execution time as well. See [21] for a report on these experiments.

5 Type-based method for testing XML query-update independence

The aim of the last section is to illustrate another use of static type analysis for XML which targets, once again, optimization but in a rather different manner. Here, the issue is to statically detect query-update independence. A query and an update are independent when the update execution does not require refreshing the query result and this for any possible input document. Detecting query-update independence is of crucial importance in many contexts: to avoid view re-materialization after update; to ensure isolation, when queries and updates are executed concurrently; to enforce access control policies. In the general case, that is for the full XQuery and XUF languages, static independence detection is undecidable [16]. This means that static analysis methods for detecting query-update independence are expected to be sound approximation as completeness is unreachable.

Recently, [16] has introduced a technique for statically detecting query-update independence: a schema is assumed available; a set of types (approximated by labels) is inferred from the query which captures the nodes accessed by the query; similarly, a set of types is inferred from the update which captures the set of nodes impacted by the update; an empty intersection of these two sets entails independence.

A path-based approach has been proposed in [40] in order to check commutativity of update expressions which is of crucial importance for optimization and view maintenance. This technique, which does not use schema information, can be adapted to deal

with queries: two sets of paths are extracted resp. from the query and from the update in the style of [46]; no overlapping of the paths in these two sets entails independence. Both techniques are effective for a wide class of XQueries and XUF and have negligible time performance.

Example 4. Let us assume that the available schema information is given by the following DTD: *doc→a* , b a→(b?,c)* c→d.*
Let us first consider the query Q_1 specified by the path doc/b together with the update U_1 given by *for $x in* $doc/a/c$ *return delete $*$x/d$. For this simple case, both methods detect independence. The schema-based method infers the sets of types $\{b\}$ and $\{a, c, d\}$ resp. for Q_1 and U_1 whereas the path-based method infers the sets of paths $\{doc/b\}$ and $\{doc/a/c, doc/d\}$.
For the second example, let us consider the query Q_1 together with the update U_2 given by *for each $x in* doc *return delete $*$x//d$. The schema-based method returns the set of types $\{a, c, d\}$ for U_2 and thus is able to detect independence. However, the path-based (schema-less) method infers, for U_2, the set of paths $\{doc//d\}$ which overlaps $\{doc/b\}$ and is unable to detect independence.
Finally, the last example considers the query Q_2 given by *for $x in* $doc/a/b$ *return* $<g>$ $\$x$ $</g>$ and the update U_1. The types associated to Q_2 by the schema-based method are $\{a, b\}$ and as it intersects $\{a, c, d\}$, independence is not spotted. However, the path associated with Q_2 by the path-based method is $doc/a/b$ and does not overlap paths associated with U_1, thus independence can be recognized.

The previous examples show that the two methods are not comparable, none of them subsume the other, and both suffer from a lack of precision for different reasons. The schema-based approach fails because the label approximation of element type is far too imprecise or not good enough. The path-based approach fails because it is too imprecise for complex axes (descendant, ancestor, horizontal axes). The approach developed in [22,23] aims at improving precision of static analysis for query-update independence. Because using schema information is often decisive (see case Q_1-U_2) as well as keeping path information (see case Q_2-U_1), this new approach consists in some hybridization of types (approximated by labels) and paths. The key idea is to use a more precise approximation (abstraction) of types for document elements. In [22,23], the notion of element type considered is that of a chain, a sequence of labels describing the path that should be navigated on to reach an element. The static analysis developed in [22,23] relies on a schema analysis radically different from that of [16,31,30,35]. The scenario of the chain-based query-update independence detection is as follows (it assumes available a schema given by a DTD or EDTD):

A set of chains is inferred from the query Q. These chains are divided into 3 kinds:
(a) used chains, intuitively pointing to document nodes used to evaluate the query Q but not involved directly in constructing the result of Q.
(b) return chains, intuitively pointing to document nodes potentially useful for constructing the elements in the result of Q.
(c) element chains, typing new elements constructed by a query (as a matter of fact, this kind of chains is important for the inference of update chains).

A set of chains is inferred from the update U which are called update chain and characterize nodes whose subtree is updated.

Checking query-update independence of Q and U relies on
(i) checking (no overlapping of) update chains against return chains, and
(ii) checking (no overlapping of) update chains against used chains

Example 5. Here, the DTD is not explicitly specified: it is the well-known bibliography schema. Let us consider the query Q given by the Xpath $//book[editor]/title$ and the update U specified by *for* $\$x$ *in* $//book$ *insert* $<author/>$ *into* $\$x$.
For the query Q, $bib.book.editor$ is a used chain: elements pointed by this path need to exist although they do not enter in the result of the query. The chain $bib.book.title\#$ is a return chain because it points to possible result elements.
For the update U, the update chain $bib.book : author$ is inferred: it is build using the used chain $bib.book$ and the element chain $author$; it is meant to capture that an update (here an insertion) may be executed below elements pointed by the chain $bib.book$ and that the new elements inserted have type $author$. Inferring the type of inserted elements is very important to increase the precision of detecting query-update independence.
Checking independence for Q and U proceeds to the following tests:
(i) is the return chain $bib.book.title\#$ a prefix of the update chain $bib.book : author$? and is the update chain $bib.book : author$ a prefix of the return chain $bib.book.title\#$? The answer to both questions here is no.
(ii) is the update chain $bib.book : author$ a prefix of the used chain $bib.book.editor$? Once again the answer to this question is no.
Therefore, independence of Q and U is inferred. Note here that both methods introduced by [16] and [40] fail to detect independence for this example.

The static analysis underlying the query-update independence presented in [22,23] is quite sophisticated. As a matter of fact, in the presence of recursive schemas, avoiding inference of infinite set of chains is a critical and rather intricate problem. An upper bound to the number of chains to be inferred has been determined, in terms of some structural properties of the query and of the update. Extensive tests of the chain-based analysis are conducted in order to validate its precision and also its time consumption.

6 Conclusion

The query optimization method, the update optimization method and the query update independence test that have been introduced in the article are all based on the static analysis of the query or/and update expression(s) and make use of the type information given by the schema of the documents. These methods show that, in the context of XML data management, schemas play an important role. For the projection-based optimization methods, type information is used for filtering fragments of the processed documents that are relevant to the queries or updates. These methods are not based on rewriting as optimization methods based on integrity constraints but still should be considered as semantic optimization. We are currently working on several directions in

order to further reduce the size of projected documents mainly by further refining the query or update expression analysis. One direction relies on making use of the kind of elementary updates occurring in a complex update expression to generate a sophisticated projector. Another direction relies on using chain rather than label for specifying the update projector, chain being a better approximation of element type than label. We are also investigating how our projection-based update optimization method can be applied to temporal XML documents in order to ensure both a compact storage of such documents and their efficient management.

Acknowledgement

It is a great pleasure for the author to thank Klaus-Dieter Schewe and Bernhard Thalheim for inviting me to present this work to the Semantics in Data and Knowledge Bases Workshop as a satellite event of ICALP 2010 in Bordeaux. I would like also to express my sincere and warm gratitude to Dario Colazzo for his leading collaboration and friendly support.
This work has been partially supported by the Codex ANR-08-DEFIS-004 project.

References

1. eXist. `http://exist.sourceforge.net/`.
2. Extensible Markup Language (XML) 1.0 (Fifth Edition). `http://www.w3.org/TR/REC-xml/`.
3. Galax. `http://www.galaxquery.org`.
4. Leo. http://leo.saclay.inria.fr/.
5. QizX Free-Engine-3.0. `http://www.xmlmind.com/qizx/free_engine.html`.
6. QizX/open. `http://www.xmlmind.com/qizx/qizxopen.shtml`.
7. SAX. http://www.saxproject.org/.
8. Saxon-ee. `http://www.saxonica.com/`.
9. XQuery 1.0: An XML Query Language. `http://www.w3.org/xquery`.
10. Xquery update facility 1.0. `http://www.w3.org/TR/2008/CR-xquery-update-10-20080801`.
11. S. Abiteboul and C. Beeri. The power of languages for the manipulation of complex values. *VLDB J.*, 4(4):727–794, 1995.
12. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.
13. A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient optimization of a class of relational expressions. *ACM Trans. Database Syst.*, 4(4):435–454, 1979.
14. M. Arenas and L. Libkin. A normal form for xml documents. *ACM Trans. Database Syst.*, 29:195–232, 2004.
15. C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *ICALP*, pages 73–85, 1981.
16. M. Benedikt and J. Cheney. Schema-based independence analysis for xml updates. *VLDB*, 2009.
17. V. Benzaken, G. Castagna, D. Colazzo, and K. Nguyen. Type-based XML projection. In *VLDB*, 2006.

18. A. Berglund, S. Boag, D. Chamberlin, M. Fernández, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0, 2005. `http://www.w3.org/TR/xpath20`.
19. N. Bidoit and D. Colazzo. Testing xml constraint satisfiability. *Electr. Notes Theor. Comput. Sci.*, 174(6):45–61, 2007.
20. N. Bidoit, D. Colazzo, N. Malla, and M. Sahakyan. Projection based optimization for xml updates. In *1st International Workshop on Schema Languages for XML (X-Schemas'09)*, 2009.
21. N. Bidoit, D. Colazzo, N. Malla, and M. Sahakyan. Projection based optimization for xml updates. In *EDBT*, 2011 (to appear).
22. N. Bidoit, D. Colazzo, and F. Ulliana. Detecting xml query-update independence. In *Workshop on Formal Methods for Web Data Trust and Security, Satellite event of IFM 2010*, 2010.
23. N. Bidoit, D. Colazzo, and F. Ulliana. Detecting xml query-update independence. In *BDA*, 2010.
24. P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes Second Edition. Technical report, World Wide Web Consortium, Oct 2004. W3C Recommendation.
25. P. Buneman. Semistructured data. In *PODS*, pages 117–121, 1997.
26. P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, and W. C. Tan. Reasoning about keys for xml. In *DBPL*, pages 133–148, 2001.
27. P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, and W. C. Tan. Keys for xml. *Computer Networks*, 39(5):473–487, 2002.
28. P. Buneman and A. Ohori. Polymorphism and type inference in database programming. *ACM Trans. Database Syst.*, 21(1):30–76, 1996.
29. L. Cardelli. A semantics of multiple inheritance. *Inf. Comput.*, 76(2/3):138–164, 1988.
30. J. Cheney. Flux: functional updates for xml. In *ICFP*, 2008.
31. J. Cheney. Regular expression subtyping for XML query and update languages. In *ESOP*, 2008.
32. J. Clark and M. Makoto. Relax NG specification. `http://www.oasis-open.org/committees/relax-ng`.
33. E. F. Codd. Further normalization of the data base relational model. *IBM Research Report, San Jose, California*, RJ909, 1971.
34. E. F. Codd. Relational completeness of data base sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California*, 1972.
35. D. Colazzo, G. Ghelli, P. Manghi, and C. Sartiani. Static analysis for path correctness of XML queries. *J. Funct. Program.*, 16(4-5), 2006.
36. J. V. den Bussche, D. V. Gucht, and S. Vansummeren. A crash course on database queries. In *PODS*, pages 143–154, 2007.
37. J. V. den Bussche and E. Waller. Polymorphic type inference for the relational algebra. *J. Comput. Syst. Sci.*, 64(3):694–718, 2002.
38. W. Fan and L. Libkin. On xml integrity constraints in the presence of dtds. *J. ACM*, 49(3):368–406, 2002.
39. W. Fan and J. Siméon. Integrity constraints for xml. *J. Comput. Syst. Sci.*, 66(1):254–291, 2003.
40. G. Ghelli, K. H. Rose, and J. Siméon. Commutativity analysis for XML updates. *ACM Trans. Database Syst.*, 33(4), 2008.
41. A. L. Hors, P. L. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. Document object model (dom). Technical report, World Wide Web Consortium, 2003. W3C Working Draft.
42. R. Jelliffe. The Schematron: An XML structure validation language using patterns in trees. `http://xml.coverpages.org/schematron.html`.

43. N. Klarlund, A. Møller, and M. I. Schwartzbach. The dsd schema language. *Autom. Softw. Eng.*, 9(3), 2002.
44. N. Klarlund, T. Schwentick, and D. Suciu. Xml: Model, schemas, types, logics, and queries. In *Logics for Emerging Applications of Databases*, pages 1–41, 2003.
45. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies (abstract). In *SIGMOD Conference*, page 152, 1979.
46. A. Marian and J. Siméon. Projecting XML documents. In *VLDB '03*, 2003.
47. M. Murata, D. Lee, and M. Mani. Taxonomy of xml schema languages using formal language theory. In *Extreme Markup Languages*, 2001.
48. L. Popa, A. Deutsch, A. Sahuguet, and V. Tannen. A chase too far? In *SIGMOD Conference*, pages 273–284, 2000.
49. D. Raggett. Assertion Grammar. `http://www.w3.org/People/Raggett/dtdgen/Docs/`.
50. R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill Science/Engineering/Math, 2002.
51. B. Thalheim. *Dependencies in Relational Databases*. Teubner Verlagsgesellschaft, Stuttgart and Leipnitz, 1991.
52. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures Second Edition. Technical report, World Wide Web Consortium, Oct 2004. W3C Recommendation.