XML data integration with identification

Antonella Poggi[1,2] and Serge Abiteboul[1]

[1] INRIA Futurs - Parc Club Orsay-University
4 rue Jean Monod, F-91893 Orsay Cedex, France
Name.Surname@inria.fr
[2] Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università di Roma "La Sapienza" - Via Salaria 113, I-00198 Roma, Italy
surname@dis.uniroma1.it

Abstract. Data integration is the problem of combining data residing at different sources, and providing the user with a virtual view, called global schema, which is independent from the model and the physical origin of the sources. Whereas many data integration systems and theoretical works have been proposed for relational data, not much investigation has been focused yet on XML data integration. Our goal is therefore to address some of its related issues. In particular, we highlight two major issues that emerge in the XML context: (i) the global schema may be characterized by a set of constraints, expressed by means of a DTD and XML integrity constraints, (ii) the concept of *node identity* requires to introduce semantic criteria to identify nodes coming from different sources. We propose a formal framework for XML data integration systems based on an expressive XML global schema, a set of XML data sources and a set of mappings specified by means of a simple tree language. Then, we define an identification function that aims at globally identifying nodes coming from different sources. Finally, we propose algorithms to answer queries under different assumptions for the mappings.

1 Introduction

Data integration is the problem of combining data residing at different sources, and providing the user with a virtual view, called global schema, which is independent from the model and the physical origin of the sources. Users query the global schema, while the system carries out the task of suitably accessing different sources and assembling the data retrieved at each source into the final answer to the query. Whereas many data integration systems [8, 10] and theoretical works [9, 6, 12] have been proposed for relational data, not much investigation has been focused yet on XML data integration. Our goal is therefore to address some of its related issues. In particular, we highlight two major issues that emerge in the XML context: (i) the global schema may be characterized by a set of constraints, expressed by means of a DTD and XML integrity constraints, (ii) the concept of *node identity* requires to introduce semantic criteria to identify nodes coming from different sources. The latter is similar to the well-studied problem of identifying objects in mediators systems [11]. However, it requires some particular solution in the context of XML data integration.

As for relational data, in order to answer a query posed over the global schema, the system needs the specification of the relationship between the sources and the global schema, which is called *mapping*. Different approaches have been proposed to specify mappings. We chose here to focus on the *Local-As-View* (LAV) approach, which

consists in characterizing the information content of the sources in terms of the global schema. An important property of mappings concerns the accuracy of the source with respect to the corresponding view. If a source provides only a subset of the data accessible from the global schema throw the corresponding view, then, we say that the mapping is *sound*. Otherwise, if the source provides exactly the corresponding view, we say that the mapping is *exact*. It is well-known that this case is more difficult to deal with. The main contributions of our work are as follows.

- First, we propose a formal framework for XML data integration systems based on (i) a global schema specified by means of a set of (simplified) DTD and a set of *XML integrity constraints* as defined in [5], (ii) a source schema specified by means of DTDs, and (iii) a set of LAV mappings specified by means of *prefix-selection-query language* that is inspired from the query language defined in [1].
- Second, we define an *identification* function, that aims at globally identifying nodes coming from different sources. As already mentioned, the need for this function is motivated by the concept of *node identity*.
- Finally, we address the query answering problem in the XML data integration setting. In particular, given the strong connection with query answering with incomplete information, we propose an approach that is reminiscent of such a context. We provide three algorithms to answer queries under the assumptions of sound, exact and mixed mappings, and study their complexity.

The paper is organized as follows. In Section 2, we illustrate XML data integration and some of its related issues by an example. In Section 3, we present the data model and the query language used in the paper. Then, the formal framework for XML data integration is introduced in Section 4, where we define the identification function. In Section 5, we introduce query answering and propose different algorithms to answer queries under the assumption of sound, exact and mixed mappings. Section 6 concludes the paper with a discussion about future works and XML data integration open issues.

Related work The only XML data integration system we are aware of, that takes into account integrity constraints, is the one presented in [3]. The authors propose the grammar AIG to specify the integration of data coming from different relational sources in a document that conforms to a DTD and satisfies a set of integrity constraints. However, in their work (i) mappings follow the *Global-As-View* (GAV) approach which has a more procedural flavor, since it characterizes the information content of the global schema in terms of the sources, (ii) the sources are relational, and (iii) whenever the retrieved data does not satisfy a constraint, the query evaluation is aborted. Closer to our work is the investigation of [2], which concerns XML data exchange. In this setting, the aim is to materialize an instance of a target schema, given an instance of a source schema, where both schemas are specified by means of DTDs. In particular, they address consistency and query answering over the target schema. However, our work considers multiple sources, whereas in data exchange the source is unique. More interestingly, no integrity constraints can be expressed over their target schema and their query language allows only for the extraction of tuples, whereas our query language extracts trees.

2 XML data integration by example

In this section, we illustrate by an example XML data integration.

Suppose that an hospital offers access to information about patients and their treatments. Information is stored in XML documents managed in different offices of the hospital, whereas users (e.g. statisticians), because of privacy and security reasons, have access to a global DTD $S_{\mathcal{G}}$ that has the following form:

$S_{\mathcal{G}}$:

```
<!ELEMENT hospital (patient+, treatment+)>
<!ELEMENT patient (SSN, name, cure*, bill?)>
<!ELEMENT treatment (trID, procedure?)>
<!ELEMENT procedure (treatment+)>
```

Following a common approach for XML data, we will consider XML documents as unordered trees, with nodes labeled with elements names. The above DTD says that the document contains data about patients and hospital treatments, where a cure is nothing but a treatment id. Moreover, a set of keys and foreign key constraints are specified over the global schema. In particular, we know that two patients cannot have the same social security number *SSN*, that two treatments cannot have the same number *trID* and that all the prescribed cures have to appear among the treatments of the hospital. Such constraints correspond respectively to two key constraints and one foreign key constraint. Finally, assume that the sources consist in the following two documents, D_1 and D_2, with the following DTDs. Mappings tell us that D_1 contains patients with a name and a social security number lower than 100000, and D_2 contains patients that paid a bill and were prescribed at least one dangerous cure (we assume that these have numbers smaller than 35).

D_1 :

```
<hospital>
  <patient>
    <name>Parker</name>
    <SSN>55577</SSN>
  </patient>
  <patient>
    <name>Rossi</name>
    <SSN>20903</SSN>
  </patient>
</hospital>
```

S_1 :

```
<!ELEMENT hospital (patient*)>
<!ELEMENT patient (name, SSN)>
```

D_2 :

```
<hospital>
  <patient>
    <SSN>55577</SSN>
  </patient>
 </hospital>
```

S_2 :

```
<!ELEMENT hospital (patient*)>
<!ELEMENT patient (SSN)>
```

Suppose now that the user asks for the following queries:

1. Find the name and the SSN for all patients having a name and a SSN, that paid a bill and that were prescribed at least one cure.
2. Does the hospital offer dangerous treatments?

Typically, in data integration systems, the goal is to find the *certain* answers, e.g. the answers that are returned by all data trees that satisfy the global schema and conform to the data at the sources. By adapting data integration terminology [9] to our setting, we call them *legal data trees*. A crucial point here is that legal data trees can be constructed by merging the source trees. We therefore need to identify nodes that should be merged, using the constraints of the global schema. Note, however, that data retrieved may not satisfy these constraints. In particular, there are two kinds of constraints violation. Data may be incomplete, i.e. it may violate constraints by not providing all data required according to the schema. Or, data retrieved may be inconsistent, i.e. it may violate

constraints by providing two elements that are "semantically" the same but cannot be merged without violating key constraints. In this paper, we will address the problem of answering queries in the presence of incomplete data, while we will assume that data does not violate key constraints. Coming back to the example, it is easy to see that the sources are consistent. Thus, the global schema constraints specification allows to answer Query 1 by returning the patient with name "Parker" and social security number "55577", since thanks to the key constraint we know that there cannot be two patients with the same SSN. Note that Query 2 can also be answered with certainty. Mappings let us actually infer that the patient named "Parker" was prescribed a dangerous cure. In addition, thanks to the foreign key constraint, we know that every cure that is prescribed to some patient is provided by the hospital.

We conclude the section by highlighting the impact of the assumption of having sound/exact mappings. Suppose that no constraints were expressed over the global schema. Under the exact mapping assumption, by inspecting the data sources, it is possible to conclude that there is only one way to merge data sources and satisfy the schema constraints. Indeed, since every patient has a name and a SSN number, we can deduce that all patients in D_2 with a SSN lower than 100000 belong also to D_1. Therefore the answer to Query 1 would be the same as in the presence of constraints, whereas no answer would be returned to Query 2, since no information is given on that portion of the global schema. On the other hand, under the assumption of sound mappings, since in the absence of constraints there could be two patients with the same SSN, both queries would return empty answers.

3 Data model and query language

In this section we introduce our data model and query language, inspired from [1].

Data trees and prefixes XML documents are represented as labeled unordered trees, called *data trees*. Given an infinite set $\mathcal{N}$ of nodes, a finite set Σ of element names (labels), and a domain $\Gamma = \Gamma' \cup \{\mathbf{0}\}$ for the data values, a *(data) tree* T over Σ is a quadruple $T = \langle t, \lambda, \nu \rangle$, where:

- t is a finite rooted tree (possibly empty) with nodes from $\mathcal{N}$;
- λ, called the *labeling function*, associates a label in Σ to each node in t; and
- ν, the *data mapping*, assigns a value in Γ to each node in t.

We call *datanodes* those nodes n of t such that $\nu(n) \neq \mathbf{0}$. Note that $\mathbf{0}$ is a special data value that represents the empty value.

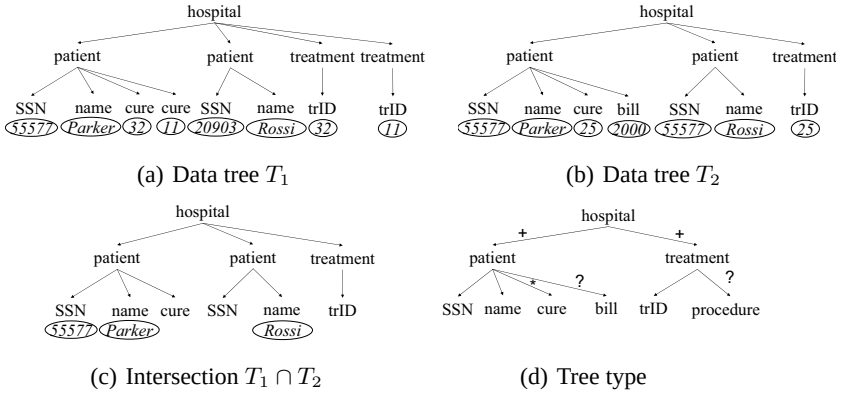
A *prefix* of $T = \langle t, \lambda, \nu \rangle$ is a data tree $T' = \langle t', \lambda', \nu' \rangle$, written $T' \leq T$, such that there exists a homomorphism h from (all) the nodes of t' to (some of) the nodes of t such that h is recursively defined as follows:

- if n' is the root of t' then $h(n')$ is defined and it is the root of t; we say that h preserves the root;
- for every node n'' that is a child of n' in t', such that $h(n')$ is defined, $h(n'')$ is defined and it is a child of $h(n')$ in t; thus h preserves the parent-child relationships;

– for every node n' in t' such that $h(n')$ is defined, $\lambda(h(n')) = \lambda'(n')$; thus h preserves the labeling;
– for every node n' in t' such that $h(n')$ is defined and $\nu(n') \neq \mathbf{0}$, $\nu(h(n')) = \nu'(n')$; thus h preserves the data mapping if and only if it maps a datanode.

Note that the empty tree, i.e. the tree that does not contain any node, denoted $T_\emptyset$, is a prefix of all data trees. Moreover, if $T' \leq T$ and $T \leq T'$, then we say that T and T' are *isomorphic*, written $T \simeq T'$. Finally, we introduce the *intersection* of two data trees. Given two data trees T_1 and T_2, their intersection, denoted $T' = T_1 \cap T_2$, is such that: (i) $T' \leq T_1, T' \leq T_2$, and (ii) for all T'' not isomorphic to T', if $T'' \leq T_1$ and $T'' \leq T_2$, then $T'' \leq T'$, i.e. T' is the maximal prefix of both T_1 and T_2.

Proposition 1. *The intersection of two data trees is unique up to tree isomorphism.*



(a) Data tree T_1 (b) Data tree T_2

(c) Intersection $T_1 \cap T_2$ (d) Tree type

Fig. 1. Data Model

Example 1. The data trees T_1 and T_2, resp. in Fig. 1(a) and 1(b), represent data about patients and treatments of an hospital. Note that only data values different from $\mathbf{0}$ are represented and are circled. In Fig. 1(c) we show the intersection $T_1 \cap T_2$. It is easy to verify that it is the maximal prefix of both T_1 and T_2.

Tree Type We call *tree type* over an alphabet Σ a simplified version of DTDs that can be represented as a triple $\langle \Sigma_\tau, r, \mu \rangle$, where Σ_τ is a set of labels, i.e. $\Sigma_\tau \subseteq \Sigma$, $r \in \Sigma_\tau$ is a special label denoting the root, and μ associates to each label $a \in \Sigma_\tau$ a *multiplicity atoms* $\mu(a)$ representing the *type* of a, i.e. the set of labels allowed for children of nodes labeled a, together with some multiplicity constraints. More precisely, $\mu(a)$ is an expression $a_1^{\omega_1} ... a_k^{\omega_k}$, where a_i are distinct labels in Σ, $\omega_i \in \{*, +, ?, 1\}$, for $i = 1, ...k$.

Given an alphabet Σ, we say that a data tree T over Σ *satisfies* a tree type $S = \langle \Sigma_\tau, r, \mu \rangle$ over Σ, noted $T \models S$, if and only if: (i) the root of T has label r, and (ii) for every node n of T such that $\lambda(n) = a$, if $\mu(a) = a_1^{\omega_1} ... a_k^{\omega_k}$, then all the children of n have labels in $\{a_1..a_k\}$ and the number of children labeled a_i is restricted as follows:

– if $\omega_i = 1$, then exactly one child of n is labeled with a_i;
– if $\omega_i = ?$, then at most one child of n is labeled with a_i;
– if $\omega_i = +$, then at least one child of n is labeled with a_i;

– if $\omega_i = *$, then no restrictions are imposed on the children of n labeled with a_i.

Given a tree type, we call *collection* a label a such that there is an occurrence of either a_i^* or a_i^+ in $\mu(a)$, for some $a_i \in \Sigma$. Moreover a_i is called *member of the collection* a.

Unary keys and foreign keys Given a tree type $S = \langle \Sigma_\tau, r, \mu \rangle$, we recall and adapt to our framework the definition of *(absolute) unary keys and foreign keys* from [5, 4]:

– Keys are assertions of the form: $a.k \rightarrow a$, where $a \in \Sigma_\tau$ and $k^1 \in \mu(a)$. The semantics of keys is the following. Given a tree T satisfying S, $T \models a.k \rightarrow a$ if and only if there does not exist two nodes n, n' of T labeled a such that their respective unique children labeled k have the same data value.
– Foreign keys are assertions of the form: $a.h_a \subseteq b.k_b$, where k_b is a key for b, $a \in \Sigma_\tau$ and $h_a^\omega \in \mu(a)$ for some ω. In this paper, we consider in particular *uniquely localizable foreign keys*, by imposing that b is such that there is a unique label path $r, l_1, .., l_s, b$ from the root to any node labeled b, where for $i = 1, .., s$, l_i is not the member of any collection. The semantics of foreign keys is the following. Given a tree T satisfying S, $T \models a.h_a \subseteq b.k_b$ if and only if for every node n of T labeled h_a that is a child of a node labeled a, there exists a node n' labeled k_b, child of a node p' labeled b, such that n and n' carry the same value (that is a key fo p').

Schema satisfaction Given a tree type $S_\mathcal{G}$, a set of keys Φ_K and a set of foreign keys Φ_{FK}, we call *schema* a triple $\mathcal{G} = \langle S_\mathcal{G}, \Phi_K, \Phi_{FK} \rangle$. Moreover, we say that a tree T *satisfies* the schema $\mathcal{G}$ if and only if $T \models S_\mathcal{G}$, $T \models \Phi_K$ and $T \models \Phi_{FK}$.

Example 2. The DTD $S_\mathcal{G}$ from Section 2 corresponds to the tree type, represented graphically in Fig. 1(d), where $r =$`hospital` and μ can be specified as follows:

`hospital` $\rightarrow$ `patient`$^+$ `treatment`$^+$
`patient` $\rightarrow$ `SSID name cure`* `bill`$^?$
`treatment` $\rightarrow$ `trID procedure`$^?$

Note that *patient* and *treatment* are both elements of the same collection *hospital*. The following sets of constraints express those mentioned in Section 2:

Φ_K : {`patient.SSN` $\rightarrow$ `patient`;
`treatment.trID` $\rightarrow$ `treatment`}
Φ_{FK} : {`patient.cure` $\subseteq$ `treatment.trID`}

The tree of Fig. 1(a) satisfies the schema $\mathcal{G} = \langle S_\mathcal{G}, \Phi_K, \Phi_{FK} \rangle$, whereas the tree of Fig. 1(b) does not since it contains two patients with the same SSN.

Prefix-selection queries Intuitively, *prefix-selection queries* (shortly referred as *ps-queries*) browse the input tree down to a certain depth starting from the root, by reading nodes with specified element names and possibly with data values satisfying selection conditions. Existential subtree patterns can also be expressed. When evaluated over a data tree T, a boolean ps-query checks for the existence of a certain tree pattern in T. A ps-query that is not boolean returns the minimal tree that is isomorphic to the set of all the nodes involved in the pattern, that are selected by the query.

Formally, a *ps-query* q *over an alphabet* Σ is a quadruple $\langle t, \lambda, cond, sel \rangle$ where:

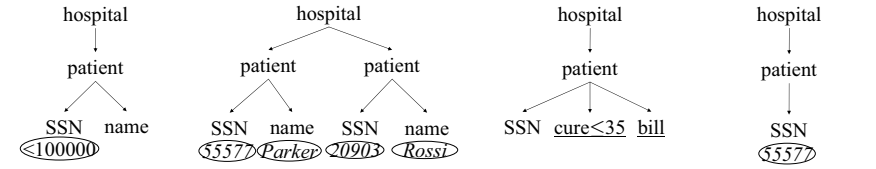
– t is a rooted tree;
– λ associates to each node a label in Σ, where sibling nodes have distinct labels.

– $cond$ is a partial function that associates to each node in t a condition c that is a boolean formula of the form $\mathbf{p_0 b_0 p_1 b_1 ... p_{m-1} b_{m-1} p_m}$, where $\mathbf{p_i}$ are predicates to be applied to datanodes values and $\mathbf{b_j}$ are boolean operators for $i = 0..m$, $m \geq 0$ and $j = 0..m-1$; for example, if $\Gamma' = \mathbb{Q}$, then predicates that can be applied to datanodes values have the form $op\ v$, where $op \in \{=, \neq, \leq, \geq, <, >\}$ and $v \in \mathbb{Q}$;
– sel is a total function that assigns to each node in t a boolean value such that if $sel(n) = false$ then $sel(n') = false$, for every children n' of n; intuitively, sel indicates whether a node is selected by the query, with the constraint that whenever n is not selected, then all the nodes of the subtree rooted at n cannot be selected.

We call *boolean ps-query* a query $q = \langle t, \lambda, cond, sel \rangle$ such that $sel_q(r_q) = false$, where r_q is the root label of t_q.

We next formalize the notion of answer to a ps-query using the auxiliary concepts of valuation and query valuation image. Given a query $q = \langle t_q, \lambda_q, cond_q, sel_q \rangle$ and a data tree $T = \langle t, \lambda, \nu \rangle$, a *valuation γ from q to T* is a homomorphism from the nodes of t_q to the nodes of t preserving the root, the parent-child relationships, the labeling and such that: for every $n_q \in t_q$, if $cond_q(n_q)$ is defined then $\nu(\gamma(n_q))$ is a datanode, i.e. $\nu(\gamma(n_q)) \neq \mathbf{0}$, and $\nu(\gamma(n_q))$ satisfies $cond_q(n_q)$. The *valuation image I of q posed over T* is the subset of nodes of T that are in the image of some valuation. We call *positive subset $P(I)$ of I* the subset of I such that for every $n \in P(I)$, there exists a valuation γ such that $sel_q(\gamma^{-1}(n)) = true$. Intuitively, $P(I)$ represents the subset of nodes of I that are selected by q.

We now define the semantics of an *answer* to a ps-query q *posed over* T, denoted as $q(T)$. If the valuation image of q posed over T is empty, then $q(T) = false$. Otherwise, $q(T)$ is a data tree such that (i) $q(T)$ is isomorphic to $P(I)$ and (ii) there does not exist a data tree T', not isomorphic to $q(T)$, such that $T' \leq q(T)$ and T' is isomorphic to $P(I)$ (i.e. $q(T)$ is the minimal tree that is isomorphic to $P(I)$). Note that if $P(I)$ is empty, then $q(T)$ is the empty tree, i.e. $q(T) = T_\emptyset$. This case occurs when q is boolean and it returns $true$.



(a) Ps-query M_1 (b) Answer to M_1 (c) Ps-query M_2 (d) Answer to M_2

Fig. 2. Querying a data tree

Proposition 2. *Given a ps-query q and a data tree T over Σ, the answer $q(T)$ is unique (up to tree isomorphism). Moreover, if $q(T) \neq false$, then $q(T)$ is the minimal prefix of T such that there exists a homomorphism h from $P(I)$ to $q(T)$ preserving parent-child relationships among nodes, labeling and data mapping.*

Example 3. Consider the queries in Fig. 2(a) and 2(c) posed over the tree of Fig. 1(a). They select respectively (i) the name and the SSN of patients having a SSN smaller than 100000, (ii) the SSN of patients that paid a bill and were prescribed at least one

dangerous cure (i.e. a cure with id lower than 35). The answers to the queries are given in Fig. 2(b) and 2(d). Note that we graphically represent an existential subtree pattern in a query by underlying the label of its root.

4 Data integration framework

In this section we first formally define a data integration system. Then we start discussing query answering by introducing an *identification function*.

4.1 Formal definition

An XML data integration system $\mathcal{I}$ can be characterized by a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where:

- The XML global schema $\mathcal{G} = \langle S_{\mathcal{G}}, \Phi_K, \Phi_{FK} \rangle$ is expressed in terms of a tree type $S_{\mathcal{G}} = \langle \Sigma_\tau, r, \mu \rangle$, a set Φ_K of key constraints and a set Φ_{FK} of uniquely localizable foreign keys. We assume that at most one key constraint is expressed for each element (e.g. Φ_K are *primary keys* [5]);
- $\mathcal{S}$ is a set of source schemas $\mathcal{S} = \{S_1, S_2, ..., S_m\}$, where S_i is a tree type, $i = 1, ..., m$; note that dealing with such kind a sources is not restrictive since we can assume that suitable wrappers are available that present the sources in this format;
- $\mathcal{M}$ is the set of (LAV) mappings between $\mathcal{G}$ and $\mathcal{S}$, one for each data source S_i in $\mathcal{S}$; they are expressions of the form: (S_i, M_i, as_i), for $i = 1, ..., m$, where $as_i \in \{sound, exact\}$ and M_i is a ps-query (not boolean) that is *coherent with* S_i, i.e. for every D_i satisfying S_i, there exists T such that $D_i \leq T$ and $M_i(T) \simeq D_i$.

Example 4. Consider the data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ that corresponds to the one discussed in Section 2. The global schema $\mathcal{G} = \langle S_{\mathcal{G}}, \Phi_K, \Phi_{FK} \rangle$ is the one of the Example 2. The source schema is $\mathcal{S} = \{S_1, S_2\}$, where S_1, S_2 correspond to the DTDs of Section 2. Finally, the mapping $\mathcal{M}$ is a set of expressions of the form: (S_i, M_i, as_i), for $i = 1, 2$, where M_i's are those of Fig. 2(a) and Fig. 2(c) and $as_i \in \{sound, exact\}$.

Given a set of data sources $\mathcal{D} = \{D_1, ..., D_m\}$ that conform to $\mathcal{S} = \{S_1, ..., S_m\}$ (i.e. $D_i \models S_i$, $i = 1, ..., m$), the semantics of a data integration system consists of all the legal data trees that conform to the schema $\mathcal{G}$ and satisfy the mappings $\mathcal{M}$. More precisely, we have the following:

$$sem(\mathcal{I}, \mathcal{D}) = \{T | T \models S_{\mathcal{G}}, T \models \Phi_K, T \models \Phi_{FK}, \\ \forall i = 1, ...m, D_i \leq M_i(T) \text{ if } as_i = sound \\ D_i \simeq M_i(T) \text{ if } as_i = exact\}$$

According to the above definition, it may happen that no legal data tree exists that belongs to $sem(\mathcal{I}, \mathcal{D})$. In this case, the setting is *inconsistent*. This may happen for the following reasons.

- The global schema specification may be *inconsistent*, i.e. there may not exist any tree that satisfies both $\mathcal{S}_{\mathcal{G}}$ and the set of constraints. It was shown in [5], that in the case of a general DTD, the problem is decidable and its complexity is NP-complete.

– A mapping may be *trivially inconsistent*, i.e. for every tree T that satisfies the global schema, $M_i(T) = T_\emptyset$. It is possible to check whether a mapping is trivially inconsistent by verifying that, given the global schema $S_\mathcal{G} = \langle \Sigma_\tau, r, \mu \rangle$ and a mapping $(S_i, M_i, as_i) \in \mathcal{M}$, with $M_i = \langle t_{q_i}, \lambda_{q_i}, cond_{q_i}, sel_{q_i} \rangle$, we have that for every $n \in t_{q_i}$: (i) if n is the root of t_{q_i}, then $\lambda_{q_i}(n) = r$, (ii) if $\lambda_{q_i}(n) = a$, all children n_i of n have distinct labels among those in $\mu(a)$. This check is clearly polynomial.
– There may be an *empty mapping*, i.e. given a source D_i, there might not exist any data tree T such that $M_i(T) \leq D_i$. This problem is also decidable. A PTIME algorithm would consist in building from M_i the query M_i' that results by ignoring the existential subtree patterns of M_i, and then checking whether $M_i'(D_i) \simeq D_i$.
– Finally, there may occur some inconsistencies among data sources and $\mathcal{G}$. In our example this would happen if two sources contain patients with the same SSN but different names.

In what follows, we will assume to deal with consistent data integration systems (note that decidability of data integration consistency problem is an open problem).

4.2 Query answering with identification

The main task of a data integration system is obviously to answer queries. Following the classical approach, we define a *certain answer* to a ps-query q posed over a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ $w.r.t.$ to a set of data sources $\mathcal{D}$, as follows:

$$q^{\mathcal{I},\mathcal{D}} = \bigcap_{T \in sem(\mathcal{I},\mathcal{D})} q(T)$$

i.e. $q^{\mathcal{I},\mathcal{D}}$ is the intersection of the answers to q over all legal data trees $w.r.t.$ $\mathcal{I}$.

Theorem 1. *Given a set of sources $\mathcal{D}$, a consistent data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and a ps-query q, $q^{\mathcal{I},\mathcal{D}}$ is the maximal data tree that is a prefix of $q(T)$, for every legal data tree T $w.r.t.$ to $\mathcal{I}$.*

Remark The certain answer to a query q posed over a data integration system $\mathcal{I}$ $w.r.t.$ to a set of data sources $\mathcal{D}$ is a data tree T such that there may not exist any T' such that $q(T') = T$. This is not surprising since by the previous theorem we have that the certain answer is the maximal *prefix of the answers* to q over all legal data trees, which only means that for every legal data tree T'', T is a prefix of the answer $q(T'')$.

To illustrate identification, let us observe the following. Suppose that no existential tree patterns were expressed in any mapping and that node ids were available that were shared among data sources. Then computing the certain answer would basically consists in merging the data sources, adding nodes to satisfy the constraints, querying the resulting tree and returning the "certain" prefix of the answer. Following this intuition, we possibly extend each data source D_i by a data source $D_i' = \langle t_i', \lambda_i', \nu_i' \rangle$ that is obtained from D_i by adding nodes whose presence can be inferred in every legal data tree from the mapping specification (S_i, M_i, as_i), where $as_i \in \{sound, complete\}$. These nodes correspond to existential tree patterns nodes in M_i. More precisely, for each leaf

$n \in D_i = \langle t_i, \lambda_i, \nu_i \rangle$ labeled a_j, we consider the node n_q of t_q such that there exists a valuation γ from t_q to D_i with $\gamma(n_q) = n$ (note that this node exists and is unique since we assumed that M_i is coherent with S_i and mappings are neither trivially inconsistent, nor empty). If m_q is a child of n_q such that $sel_q(m_q) = false$, then we recursively proceed as follows. For every node m'_q in the subtree rooted at m_q, a node m is added in $D_i = \langle t_i, \lambda_i, \nu_i \rangle$ such that we can extend γ by defining $\gamma(m'_q) = m$, where:

- m is child of the node n of t'_i such that $\gamma^{-1}(n)$ is defined and it is the parent of m'_q;
- $\lambda'_i(m) = \lambda_q(m_q)$;
- if $cond_q(m_q)$ is defined, then $\nu'_i(m) = v_s$ where v_s is a fresh Skolem constant such that $cond_q(m_q)$ is satisfied.

Next, we define the *Identification* function whose aim is to obtain from each extended data source D'_i a new data source, called *identified data source*, whose nodes have global ids that depend on $\mathcal{G}$, such that two nodes have the same global id only if they are merged in every legal data tree. In order to introduce the identification, we start by recursively defining the domain $\mathcal{N}^I$ of global ids:

- $\epsilon \in \mathcal{N}^I$;
- if $\mathbf{n} \in \mathcal{N}^I$, then $\mathbf{n}.a_i[.\gamma_i] \in \mathcal{N}^I$, where $a_i \in \Sigma$ and γ_i is an optional value in $\bar{\Gamma} = \Gamma \cup \mathcal{V}^S$, where $\mathcal{V}^S$ is a set of Skolem constants.

Finally, $Id(D)$ is obtained by recursively associating to each node n in $D'_i = \langle t'_i, \lambda'_i, \nu'_i \rangle$ a global id $\mathbf{id_n}$ in $\mathcal{N}^I$:

- if n is the root of t'_i, then $\mathbf{id_n} = \epsilon$;
- if n labeled a_j is child of a node p labeled a, $\mathbf{id_n} = \mathbf{id_p}.a_j[.\gamma]$ where γ is an optional value appearing if:
  - either there exists $a_j.k \rightarrow a_j \in \Phi_K$; then if n has a child m labeled k, then $\gamma = \nu(m)$, otherwise $\gamma = v_s$ where v_s is a fresh constant in $\mathcal{V}^S$;
  - or $a_j^{\omega_j} \in \mu(a)$, where $\omega_j \in \{+, *\}$; then $\gamma = v_s$, with v_s fresh constant in $\mathcal{V}^S$.

Note that, by an abuse of notation, we denote $Id(D_i)$ the data source obtained by first extending the original data source and then identifying nodes as described. $Id(D_i)$ is such that all its nodes have a global id. If $\mathbf{id_n}$ does not contain any Skolem constant, we say that n is uniquely identified. In the following example, we illustrate identification.

Example 5. Given the data integration system of Example 4 and the source D_1 given in Fig. 2(d), $Id(D_1)$ is represented in Fig. 3, where the labels of nodes added by the identification are boxed, the global ids are marked in bold and γ_i represent Skolem constants in $\mathcal{V}^S$, for $i = 1, 2$. Note that all nodes are uniquely identified, except for the node labeled `cure`. Moreover, γ_1 represents a data value lower than 35.

By identifying data sources, clearly, two nodes are assigned the same global id only if they are merged in every legal data tree. Moreover, the data sources extension does not modify the sources content that is mapped to every legal data tree. It therefore does not affect certain answers. Indeed, it is straightforward to prove the following theorem.

Theorem 2. *Given a set of data sources $\mathcal{D}$ and a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, the following holds:*

$$q^{\mathcal{I},\mathcal{D}} = \bigcap_{T \in sem(\mathcal{I},\mathcal{D})} q(T) = \bigcap_{T \in sem(\mathcal{I}, Id(\mathcal{D}))} q(T).$$

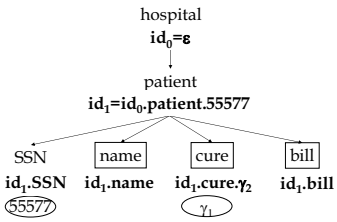


Fig. 3. Identified tree $Id(\mathcal{D})$

From now on, the previous theorem will let us consider $sem(\mathcal{I}, Id(\mathcal{D}))$ rather than $sem(\mathcal{I}, \mathcal{D})$.

5 Query answering algorithms

In this section, we provide three algorithms that use identification to answer ps-queries over $\mathcal{I}$, under the assumption of sound, exact and mixed mappings. All proposed algorithms follow an approach that is typical in the presence of incomplete information. This is not surprising since it is well-known [6] that LAV data integration query answering is strongly related to the problem of querying an incomplete database. Indeed, data sources provide only partial information on legal data trees. Thus, our algorithms are all based on the idea of constructing a weak representation system $\mathbf{T}$ [7], to represent all legal trees (i.e. $sem(\mathbf{T}) = sem(\mathcal{I}, Id(\mathcal{D}))$), such that for each $\mathbf{T}$ and each ps-query q there exists a representation $\mathbf{q}(\mathbf{T})$ such that $\bigcap\{T|T \in sem(\mathbf{q}(\mathbf{T}))\} = \bigcap\{q(T)|T \in sem(\mathbf{T})\}$. It follows that the complexity is given by (i) the complexity of computing $\mathbf{T}$, (ii) the complexity of constructing $\mathbf{q}(\mathbf{T})$, and (iii) the complexity of computing the intersection of answers represented by $\mathbf{q}(\mathbf{T})$.

5.1 Query answering under sound mappings

Before introducing the algorithm for query answering, we highlight that, as mentioned earlier, it is based on the idea of building a weak representation system $\mathbf{T}$. Because of lack of space, instead of introducing formally $\mathbf{T}$, we intuitively present it as a special tree with values in $\bar{\Gamma}$. In particular, $\mathbf{T}$ may have Skolems as data values that are constrained to satisfy conditions similar to those expressed by ps-queries (note that this may happen, for example, when nodes are added in order to satisfy existential subtree patterns in the mappings, or a constraint of the schema). The valuation of a query q over $\mathbf{T}$ has to be modified accordingly. In particular, $\mathbf{q}(\mathbf{T})$ may contain Skolem constants for data values, since it has to be equivalent to $\mathbf{T}$ from the point of view of certain answers to q.

Given a query q, a system $\mathcal{I}$ and a set of sources $\mathcal{D} = \{D_1, ..., D_m\}$, our algorithm for query answering under the assumption of sound mappings proceeds as follows.

1. We compute $Id(\mathcal{D})$ $w.r.t.$ to $\mathcal{G}$ and obtain a data tree with global ids and with data values in $\bar{\Gamma}$. However, this tree may contain nodes that are semantically equivalent, i.e. they represent the same node in every legal data tree, but have different node identifiers. In particular, this may happen when $Id(D_i)$ and $Id(D_j)$ contain resp. nodes n_i, n_j labeled a uniquely identified by the same id $\mathbf{id_n}$ and nodes m_i, m_j, labeled b, resp. children of n_i, n_j, that are identified with different global ids, whereas

according to $S_{\mathcal{G}}$, nodes labeled a should have at most one child labeled b. To simplify, suppose that they are not datanodes. If at least one among m_i, m_j, say m_i, is not uniquely identified, then the sources are consistent and we say that m_i, m_j can be *unified*, by replacing the global id of n_i with the id of n_j. We then obtain the *retrieved global data tree* $w.r.t.$ to $\mathcal{D}$, denoted $ret(\mathcal{I}, \mathcal{D})$. It is possible to prove that if the setting is consistent, then $ret(\mathcal{I}, \mathcal{D})$ is such that $ret(\mathcal{I}, \mathcal{D}) \leq T$, for every legal data tree T.

2. We compute the representation system $\mathbf{T} = \langle t^*, \lambda^*, \nu^* \rangle$ for all legal data trees, by adding nodes to $ret(\mathcal{I}, \mathcal{D})$ in order to satisfy $\mathcal{G}$. More precisely, we proceed by applying the following rules:
   (a) For each p labeled a, if $a_i^{\omega_i} \in \mu(a)$ where $\omega_i \in \{1, +\}$ and n' has not any child labeled a_i, then we add to $\mathbf{T}$ the child n of p, with $\lambda^*(n) = a_i$ and $\nu^*(n) = \mathbf{0}$. If $a_i.k \rightarrow a_i \in \Phi_K$ or $\omega_i = +$, then $\mathbf{id_n} = \mathbf{id_p}.a_i.\gamma_s$ where γ_s is a fresh constant from $\mathcal{V}^s$, otherwise $\mathbf{id_n} = \mathbf{id_p}.a_i$.
   (b) For each m_a labeled h_a, child of n_a labeled a, if $a.h_a \subseteq b.k_b \in \Phi_{FK}$, and there is not any node labeled b with key value $\nu(m_a)$, then we add a set of nodes, one node n' for each label l that occurs from the root to the parent of the node labeled b in $S_{\mathcal{G}}$, where $\lambda^*(n') = l$ and $\nu^*(n') = \mathbf{0}$, so that the tree satisfies the global schema. Note that since the foreign key constraints are uniquely localizable, all these nodes are uniquely identified and therefore their global ids depend only on $\mathcal{G}$. Suppose that p is the last node that is added, and that its global id is $\mathbf{id_p}$. Then we add the node n_b child of p, with global id $\mathbf{id_{n_b}} = \mathbf{id_p}.\nu(m_a)$ and such that $\lambda^*(n_b) = b$ and $\nu^*(n_b) = \mathbf{0}$. Moreover, we add the child m_b of n_b, with global id $\mathbf{id_{m_b}} = \mathbf{id_{n_b}}.k_b$ such that $\lambda^*(m_b) = k_b$ and $\nu^*(m_b) = \nu(m_a)$.

   Intuitively, this step corresponds to computing the well-known technique of the *Chase* over $ret(\mathcal{I}, \mathcal{D})$. Since the Chase may not stop and lead to an infinite data tree, we proceeds as long as the algorithm adds nodes (that are required by the schema) that have either the form $\mathbf{id_n} = \mathbf{id_p}.\mathbf{a}$ or the form $\mathbf{id_n} = \mathbf{id_p}.\mathbf{a}.\gamma_\mathbf{s}$ where $a \in \Sigma_\tau$, $\gamma_s \in \mathcal{V}^s$ is a Skolem constant and there is not any node with global id $\mathbf{id_n} = \mathbf{id_p}.\mathbf{a}.\gamma'_\mathbf{s}$ where $\gamma'_s \in \mathcal{V}^s, \gamma'_s \neq \gamma_s$.
3. We compute $\mathbf{q}(\mathbf{T})$ and return its certain prefix $\bar{T} = \langle \bar{t}, \bar{\lambda}, \bar{\nu} \rangle$. Note in particular that for every n in $\mathbf{q}(\mathbf{T})$ such that $\nu^*(n)$ is a Skolem, we set $\bar{\nu}(n) = \mathbf{0}$.

Claim. Given a consistent data integration system, the above algorithm terminates.

Theorem 3. *Given a consistent data integration system $\mathcal{I}$ with sound mappings, a set of sources and a ps-query q, $q^{\mathcal{I},\mathcal{D}} = \bar{T}$, where $\bar{T}$ is computed as above.*

Complexity Let us discuss the computational complexity of the above algorithm. In particular, we focus on data complexity, which refers to the size of the set of data sources $\mathcal{D}$. It is easy to see that the construction of the data tree $\mathbf{T}$ is polynomial in the size of $\mathcal{D}$. Moreover, the size of $\mathbf{T}$ is polynomial. Then, since the construction of $\mathbf{q}(\mathbf{T})$ is polynomial in the size of $\mathbf{T}$, we have that our algorithm is PTIME.

5.2 Query answering under exact mappings

Suppose now that mappings are exact. As for the representation of legal data trees, we can use incomplete trees of [1], known to be a strong representation system for ps-

queries. However, we have to deal with three major differences $w.r.t.$ [1]. The first is that persistent node ids are not available in our setting. As for global ids obtained by first identifying the sources and then computing possible unifications, there still may occur two nodes with different global ids that represent the same node in every legal data tree. Recall the example from the end of Section 2. Let us call n_1, n_2 the two nodes labeled `patient` belonging to D_1, n_3 the node with the same label belonging to D_2. Without keys, n_1, n_2, n_3 would be assigned different global ids. However, as already discussed, under exact mappings, we can conclude that n_1 and n_3 are the same in every legal tree. Thus, in order to correctly identify source nodes, view definitions should also be taken into account. Of course, this would notably increase the query answering complexity.

Therefore, we introduce the *Visible Keys Restriction (VKR)* for $\mathcal{I}$:

- For every element a, member of a collection of $S_\mathcal{G}$, there exists $a.k \rightarrow a \in \Phi_K$.
- For every view M_i such that $(S_i, M_i, exact) \in \mathcal{M}$, M_i is such that whenever it selects an element with a key, it also selects its key.

This ensures that all nodes can be uniquely identified. Then, by identifying the sources, we reduce our setting to the case of [1] where global ids play the role of persistent ids.

The second major difference $w.r.t.$ [1] is the ps-query language. In this paper, we extend ps-queries of [1] in order to make them express existential subtree patterns. It is possible to prove that such an extension maintains all the good properties of ps-queries. In particular, incomplete trees (whose construction is accordingly modified) are still a strong representation system for our query language.

The third major difference $w.r.t.$ [1] is the presence of foreign keys. Intuitively, we introduce additional sources that contain the data required to satisfy foreign keys. More precisely, given a foreign key $a.h_a \subseteq b.k_b \in \Phi_{FK}$, for each sequence of labels from the root to nodes labeled h_a in $Id(\mathcal{D})$, we introduce a source D_{m+j} containing all nodes m_a in $Id(\mathcal{D})$ labeled h_a characterized by that sequence of labels, together with their ancestors. Moreover, the source will contain all the nodes from the root to nodes n_b, m_b with respective labels b, k_b, such that for each node m_a there is a node n_b with key value $\nu(m_b) = \nu(m_a)$. Note that, after identification, under the assumption of uniquely localizable foreign keys, all the ancestors of nodes labeled b are uniquely identified.

Under VKR assumption, the algorithm with exact mappings proceeds as follows.

1. We compute $Id(\mathcal{D})$.
2. To guarantee the satisfaction of foreign key constraints, for every $a.h_a \subseteq b.k_b \in \Phi_{FK}$ and for every different sequence of labels from the root to nodes $n \in Id(\mathcal{D})$ labeled a, we build a data source and the corresponding view definition as described above. We call $D_{m+1}, ...D_{m+k}$ the new data sources. Then we add the corresponding exact mappings to $\mathcal{M}$, for every $j = m+1, ...m+k$.
3. We compute the incomplete tree $\mathbf{T_i}$ s.t. $sem(\mathbf{T_i}) = \{T | M_i(T) \simeq Id(D_i)\}$, for $i = 1, .., m$ and $\mathbf{T_j}$ s.t. $sem(\mathbf{T_j}) = \{T | M_j(T) \simeq Id(D_j)\}$, for $j = m+1, .., m+k$.
4. We compute the incomplete tree $\mathbf{T'}$ such that $sem(\mathbf{T'}) = \bigcap_{i \in \{1,..,m+k\}} sem(\mathbf{T_i})$.
5. In order to take into account the tree type $S_\mathcal{G}$, we compute the incomplete tree $\mathbf{T}$ such that $sem(\mathbf{T}) = sem(\mathbf{T'}) \cap sem(S_\mathcal{G})$.
6. We query the incomplete tree $\mathbf{T}$ and obtain the representation $\mathbf{q}(\mathbf{T})$.

Theorem 4. *Given a consistent $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, with $\mathcal{M}$ all exact, a set of sources $\mathcal{D}$ for $\mathcal{I}$ and a ps-query q, under the VKR assumption, $q^{\mathcal{I},\mathcal{D}} = \bigcap \{T | T \in sem(\mathbf{q}(\mathbf{T}))\}$.*

Complexity In [1], it was shown that computing $\mathbf{q}(\mathbf{T})$ is PTIME in data complexity. Moreover, checking whether $\bar{T}$ is a certain prefix of $\mathbf{q}(\mathbf{T})$ is PTIME in the size of $q(\mathbf{T})$, itself PTIME. We strongly conjecture that checking whether $\bar{T}$ is the maximal certain prefix is also PTIME. On the other hand, it was also shown in [1] that the problem of deciding whether $\bar{T}$ is a certain prefix of $\mathbf{q}(\mathbf{T})$ is NP-complete in the sequence of ps-queries. We therefore conjecture that checking whether $\bar{T}$ is a certain answer of q over $\mathcal{I}$ is NP-complete in the number of data sources.

5.3 Query answering under mixed mappings

We now present an algorithm to answer queries under sound and/or exact mappings. Suppose that a different color C_i is associated to each source D_i characterized by a sound mapping. The idea is to reduce the query answering algorithm under mixed mappings to query answering with exact mappings. Suppose that we have a collection of nodes labeled a_i in $S_\mathcal{G}$ and a source D_i provides some sound information about this collection. We can abstractly consider D_i as the source providing exactly the nodes with label a_i and color C_i. This requires to add the information about the color to each node of the collection. Then, under the VKR assumption, we are able to answer queries.

1. We compute the extended tree type $\hat{\mathcal{S}_\mathcal{G}} = \langle \hat{\Sigma_\tau}, r, \hat{\mu} \rangle$ by modifying $S_\mathcal{G}$ so that $\hat{\Sigma_T} = \Sigma_T \cup \mathtt{C}$, and $\hat{\mu}$ is defined as follows:

$$\forall a \in \hat{\Sigma}, \hat{\mu}(a) = \begin{cases} \mu(a)\mathtt{C}^* & \text{if } a \text{ is a member of some collection of } \mathcal{S}_\mathcal{G} \\ \mu(a) & \text{otherwise} \end{cases}$$

2. For every sound mapping, we build $\hat{M_i} = \langle \hat{t_i}, \hat{\lambda_i}, \hat{cond_i}, \hat{sel_i} \rangle$ by modifying $M_i = \langle t_i, \lambda_i, cond_i, sel_i \rangle$ so that for every node $n \in t_i$ such that $\hat{\lambda}_i(n) = a$, where a is a member of some collection in $\mathcal{S}_\mathcal{G}$, n has a child m labeled $\mathtt{C}$, such that $\hat{cond_i}(m) = " = C_i"$, where C_i is the color of D_i.
3. For every data source D_i with color C_i, characterized by a sound mapping, we build the data source $\hat{D_i} = \langle \hat{t_i}, \hat{\lambda_i}, \hat{\nu_i} \rangle$ by adding to every node $n \in D_i$, labeled with a member of a collection, a child data node m labeled with $\mathtt{C}$ such that $\hat{\nu_i}(m) = C_i$.
4. We apply the algorithm described in the previous section and obtain the incomplete tree $\hat{\mathbf{T}}$. Then we query it and obtain the representation $\mathbf{q}(\hat{\mathbf{T}})$.

Theorem 5. *Given a consistent $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, with $\mathcal{M}$ mixed, a set of sources $\mathcal{D}$ for $\mathcal{I}$ and a ps-query q, under the VKR assumption, $q^{\mathcal{I},\mathcal{D}} = \bigcap\{T | T \in sem(\mathbf{q}(\hat{\mathbf{T}}))\}$.*

Complexity Obviously, the complexity is the same as in the previous case.

6 Discussion and future works

We have presented a formal framework for XML data integration, based on an expressive global schema specified by means of a simplified DTD and XML keys and foreign keys. We have shown how to address the issue of identifying nodes coming from different sources. Then, we have proposed query answering algorithms assuming that mappings are sound and/or exact. It turns out that under sound mappings assumption, we are able to answer queries in polynomial time in data complexity. In the exact case, in order to limit the complexity, we have introduced an extra condition, namely

the *Visible Key Restriction (VKR)*. We strongly conjecture that, under this assumption, the query answering problem is NP in the number of views. This apparent loss allows to gain very much in terms of expressivity of the representation system that we use to answer queries. In particular, we may ask for *possible answers*. We may also extract a precise description about missing information as shown in [1]. This information may be used to guide our system in finding additional data sources to answer queries. Moreover, we showed how to incorporate information about the data origin. This may be very useful to optimize query answering. In fact, our algorithms are all naive, in that they proceed by first constructing a representation of all legal data trees and then by querying it. The next step is to propose more efficient algorithms that do not need to build such representation but only the portion of interest with respect to the query.

In the future, we plan to follow several other research directions. In particular, we will study more carefully the complexity of query answering in the exact case. Then, we plan to consider more expressive query languages for the specification of the mappings as well as for querying the system. An orthogonal issue would concern data sources inconsistencies. To this aim, repair techniques for XML data would be required.

Acknowledgements We are very grateful to Maurizio Lenzerini and Diego Calvanese for the helpful discussions that lead to the revision of the first draft, to Ioana Manolescu and Bogdan Cautis for having helped us by carefully reading the paper and giving precious feedbacks, and to Luc Segoufin for many interesting discussions about the topic. Finally we would also like to thank the referees for several very useful comments.

References

1. S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying xml with incomplete information. In *Proc. of ACM PODS*, 2001.
2. M. Arenas and L. Libkin. Xml data exchange: Consistency and query answering. In *Proc. of ACM PODS*, 2005. To Appear.
3. M. Benedikt, C. Chan, W. Fan, J. Freire, and R. Rastogi. Capturing both Types and Constraints in Data Integration. In *Proc. of ACM SIGMOD*, 2003.
4. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. In *Proc. of the Int. WWW Conf.*, pages 201–210, 2001.
5. W. Fan and L. Libkin. On XML Integrity Constraints in the Presence of DTDs. *J. ACM*, 49(3):368–406, 2002.
6. A.Y. Halevy. Answering queries using views: A survey. *Very Large Database J.*, 10(4):270–294, 2001.
7. J. Imielinski and W. Lipski. Incomplete information in relational databases. *JACM*, 31(4), 1984.
8. T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The information manifold. In C. Knoblock and A. Levy, editors, *Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California, 1995.
9. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of ACM PODS*, 2002.
10. C. Li, R. Yerneni, V. Vassalos, H. Garcia-Molina, Y. Papakonstantinou, J. D. Ullman, and M. Valiveti. Capability based mediation in TSIMMIS. In *Proc. of ACM SIGMOD*, 1998.
11. Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proc. of VLDB*, 1996.
12. Jeffrey D. Ullman. Information integration using logical views. In *Proc. of Intl. Conf. on Database Theory*, 1997.