XML Schema Mappings in the Presence of Key Constraints and Value Dependencies [⋆]

Tadeusz Pankowski[1,2], Jolanta Cybulka[1], and Adam Meissner[1]

[1] Institute of Control and Information Engineering,
Poznań University of Technology, Poland
[2] Faculty of Mathematics and Computer Science,
Adam Mickiewicz University, Poznań, Poland
tadeusz.pankowski@put.poznan.pl

Abstract. Schema mappings play a central role in both data integration and data exchange, and are understood as high-level specifications describing the relationships between data schemas. Based on these specifications, data structured under a source schema can be transformed into data structured under a target schema. During the transformation some structural constraints, both context-free (the structure) and contextual (e.g. keys and value dependencies) should be taken into account. In this work, we present a new formalism for the schema mapping specification. We propose a new class of tree-pattern formulas in order to extend semantics of XML schema mappings by specification of key constraints and value dependencies. We discuss foundations of the method and propose a key-preserving transformation algorithm.

1 Introduction

Many problems in information integration and data exchange need *schema mappings*, which are high-level specifications describing the relationships between source and target data schemas. A schema mapping over a pair of schemas expresses a relation on the set of instances of these two schemas. The mapping should produce an instance of the target schema for a given instance of the source schema. The target instance should correctly represent the information contained in the source instance under the constraints imposed by the target schema [3, 11, 19, 29]. The problem of schema mapping arises in many areas of data management systems. Recently, this problem has received considerable attention in the context of data exchange [3, 10], data integration [16], schema evolution [18], P2P databases [5, 20], life science databases [21] or e-commerce [6], where data comes from many different sources with different schemas. In data management [18] such operations on mappings as composition [17, 11, 19] and inversion [9] has been considered.

Schema mappings between relational schemas are usually defined by means of the *source-to-target dependencies* [1, 12, 19]. Recently, this method was adapted

[⋆] The work was supported in part by the Polish Ministry of Science and Higher Education under Grant N516 015 31/1553.

to XML data and *tree-patterns* are used to capture the context-free structure of XML trees [3]. However, the approach does not take (contextual) key constraints into account.

In this paper we assume that the definition of a schema is specified in XSD (XML Schema Definition [27]) language, and except for context-free constraints the following three classes of contextual constraints are taken into account: *keys*, *key references*, and *value dependencies*.

1. *Key constraints* state that a subtree is uniquely identified by a tuple of values of key paths [4, 27]. They are specified within the `<xs:key>` elements.
2. *Keyref constraints* assert a correspondence of the tuples of values resulting from evaluating a referencing key (a foreign key) with those of the referenced key [27]. They are specified within the `<xs:keyref>` elements.
3. *Value dependency constraints* impose that a text value of a text node depends on a tuple of values of other nodes (resembling functional dependencies in relational databases [1]). The value dependencies are declared in the `<xs:valdep>` section of XSD (this section can be included in `<xs:annotation>` element of XSD).

We introduce *key-pattern* formulas which are intended to represent keys defined within XSD, and are used to control mapping execution. It makes the process of XML data mappings and transformations semantically richer. We propose algorithms that construct an intended key-preserving target instance, we also formulate a theorem that is the basis for the presented method. In our approach, we distinguish among three kinds of mappings: *automappings* which are automatically generated from XSD specifications and capture both context-free and contextual properties; *c-mappings* that express correspondences between source and target schemas; t-mappings which are derived from auto- and c-mappings and define key-preserving transformations. The advantage of this approach is that the management of these mappings and the reasoning over them can be carried out independently.

The structure of the paper is as follows. In Section 2 formal definitions concerning XML trees and XML schemas are given and a running example is presented. The fundamentals of schema mappings are reviewed in Section 3. In this section we also characterize the existing approaches. In Section 4 we propose a formalism for defining key-preserving XML schema mappings. The theorem and the algorithm concerning the mapping execution are presented in Section 5. Section 6 concludes the work and formulates directions of the future research.

2 XML trees and XML schemas

We consider XML data as a node-labeled unranked and ordered tree (XML tree) [28]. For simplicity, attribute nodes are represented by text nodes.

Let $L \subset Lab$ be a finite set of labels, Str be a set of string values (with the distinguished *null* values $\perp, \perp_1, \perp_2, ...$ in Str), and DId be a set of document identifiers (e.g. URI addresses).

Definition 1. *An XML tree I is a tuple $(r, N^e, N^t, \leq, child, \lambda, \nu, \sigma)$, where:*

- *r is a distinguished root node, N^e is a finite set of element nodes, and N^t is a finite set of text nodes;*
- *$\leq$ – a total ordering relation on the set of nodes;*
- *$child \subseteq (\{r\} \cup N^e) \times (N^e \cup N^t)$ – an acyclic binary relation introducing tree structure into (r, N^e, N^t) such that the root has only one child (this child is the top element) and each element node must have a child;*
- *$\lambda : N^e \to L$ – a function labeling element nodes (the label is the type of the node);*
- *$\nu : N^t \to Str$ – a function labeling text nodes with their text values;*
- *$\sigma : \{r\} \to DId$ – a function that assigns a document identifier to the root.* □

The XML trees will be considered as instances of XML schemas. We will restrict ourselves to context-free and contextual (key) constraints specified by an XML schema. The context-free constraints are given by regular expressions describing types of children for any element node type. Keys specify how subtrees are identified within an XML tree by means of text values of *key paths.*

A *path* P is defined by the grammar $P ::= /l \mid //l \mid l \mid P/l \mid P//l$, where: $/l$ denotes a set of children of type l of the root, $//l$ denotes all descendent nodes of type l of the root, P/l denotes a set of nodes of type l which are children of the nodes denoted by P, $P//l$ denotes all "descendent-or-self" nodes of type l of the nodes denoted by P.

We assume that there is a key for any node type. Following [4], we assume the following definition of a key for XML:

Definition 2. *A key is an expression of the form $(P, (P', (P_1, ..., P_k)))$, where every $P/P'/P_i$ is a valid path. The path P is the context path, P' is the target path, and $P_1, ..., P_k$ are key paths of the key.* □

Definition 3. *An XML schema S over a set L of labels is a tuple*

$$S = (top, ContFree, KeyDep, KeyrefDep, ValDep),$$

where:

- *top is the the distinguished label of the top (outermost) element;*
- *$ConFree$ is a function from L to regular expressions over $L - \{top\}$ defined by the grammar $e ::= \epsilon \mid l \mid e|e \mid e, e \mid e? \mid e+ \mid e*$;*
- *$KeyDep$ assigns a key $(P, (l, (P_1, ..., P_k)))$ to any label $l \in L - \{top\}$.*
- *$KeyRefDep$ assigns referential constraints, i.e. expressions of the form, $(P, l, (P_1, ..., P_k))$ **ref** $KeyDep(l')$, to some labels in L; $l, l' \in L - \{top\}$. The expression defines a foreign key $(P, l, (P_1, ..., P_k))$ that refers to a primary key $KeyDep(l')$.*
- *$ValDep$ assigns a set of value dependencies to some labels in L. A value dependency is an expression of the form $(P/l, P') = f(P/l/P_1, ..., P/l/P_m)$.*

Definition 4. *An XML tree $I = (r, N^e, N^t, \leq, child, \lambda, \nu, \sigma)$ conforms to the XML schema $S = (top, ConFree, KeyDep, KeyRefDep, ValDep)$, if:*

1. *$\sigma(r)$ is defined, and for any text node $n \in N^t$, $\nu(n)$ is defined.*
2. *If $(r, n) \in child$, then $\lambda(n) = top$.*
3. *For any node n in N^e with children $(n_1, ..., n_m)$ such that $n_1 < ... < n_m$, if $\lambda(n) = l$, then the sequence $\lambda(n_1)...\lambda(n_m)$ is a word of the language defined by the regular expression $ConFree(l)$.*
4. *If $KeyDep(l) = (P, (l, (P_1, ..., P_k)))$, then each subtree rooted in a node n of type l in an context denoted by the context path P, is uniquely identified by a tuple of text values $(v_1, ..., v_k)$ of the key paths $(P_1, ..., P_k)$, $k \geq 0$. For $k = 0$, $(P, (l, ()))$ indicates that n is unconditionally unique in the context of P.*
5. *If $KeyRefDep(l) = (P, l, (P_1, ..., P_k))$* **ref** *$(P', (l', (P'_1, ..., P'_k)))$, then for any tuple $(v_1, ..., v_k)$ of values such that $P[l[P_1 = v_1 \wedge \cdots \wedge P_k = v_k]]$ there exists exactly one node $n \in [\![P'/l']\!]$ such that the tuple $(P'_1, ..., P'_k)$ of key paths on n has the value equal to $(v_1, ..., v_k)$.*
6. *If $(P/l, P') = f(P/l/P_1, ..., P/l/P_n) \in ValDep(l)$, then text values of the path $P/l/P'$ are functionally dependent on the set of tuples of text values determined by $(P/l/P_1, ..., P/l/P_n)$. We use the name f to distinguish two different dependencies having the same set of determining paths.* $\square$

An XML tree I is called an *instance* of a schema S, denoted $I \models S$, iff conforms to S, i.e. satisfies all the constraints imposed by S.

Example 1. Sample XML trees and XML schema trees are given in Fig. 1. The XML trees in Fig. 1 represent the bibliographical data. The node labels are as follows: paper (P) and title (T) of the paper; author (A), name (N) and the affiliation (U) of the author; year (Y) of publication and the conference (C) where the paper was presented; R and K are used to join authors with their papers.

If we restrict ourselves to the context-free structure only, then definitions of S_1, S_2 and S_3 can be given as DTDs in Fig. 2. In order to specify contextual constraints, we will use the schema definition language XSD. In Fig. 4 the complete XSD definitions for S_1 and fragments for S_2 and S_3 are given. It can be seen that the XML tree I_1 is an instance of S_1, whereas I_2 and I'_2 are instances of S_2 with respect to the DTDs D_1 and D_2, respectively. Moreover, I_1 and I_2 are instances of schemas specified by XSDs SD_1 and SD_2, since they satisfy keys, respectively:

$$\begin{array}{l}(S1, (P, (T))), (S1/P, (T, ())), (S1/P, (A, (N))),\\ (S1/P/A, (N, ())), (S1/P/A, (U, ())),\end{array} \tag{1}$$

and

$$\begin{array}{l}(S2, (A, (N))), (S2/A, (N, ())), (S2/A, (U, ())),\\ (S2/A, (P, (T))), (S2/A/P, (T, ())), (S2/A/P, (Y, ())).\end{array} \tag{2}$$

Note, that I'_2 satisfies the context-free constraints specified in SD_2 but does not satisfy keys. In fact, it satisfies the following key:

$$\begin{array}{l}(S2, (A, (N, P/T))), (S2/A, (N, ())), (S2/A, (U, ())),\\ (S2/A, (P, (T))), (S2/A/P, (T, ())), (S2/A/P, (Y, ())).\end{array} \tag{3}$$

For example, each subtree in I_2' denoted by $/S2/A$ is uniquely identified by a pair of text values of the pair of key paths $(N, P/T)$. In Figure 3 there are constraints assigned to elements of the schema S_3. □

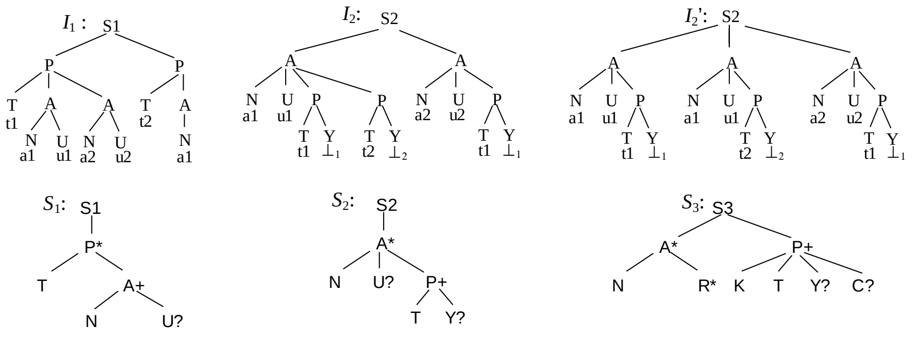

Fig. 1. Sample XML trees and XML schema trees

```
D1 : <!DOCTYPE S1 [              D2 : <!DOCTYPE S2 [              D3 : <!DOCTYPE S3 [
     <!ELEMENT S1 (P*)>               <!ELEMENT S2 (A*)>               <!ELEMENT S3 (A*, P+)>
     <!ELEMENT P (T, A+)>             <!ELEMENT A (N, U, P+)>          <!ELEMENT A (N, R*)>
     <!ELEMENT T (#PCDATA)>           <!ELEMENT N (#PCDATA)>           <!ELEMENT N (#PCDATA)>
     <!ELEMENT A (N, U?)>             <!ELEMENT U (#PCDATA)>           <!ELEMENT R (#PCDATA)>
     <!ELEMENT N (#PCDATA)>           <!ELEMENT P (T, Y?)>             <!ELEMENT P (K, T, Y?, C?)>
     <!ELEMENT U (#PCDATA)>]>         <!ELEMENT T (#PCDATA)>           <!ELEMENT K (#PCDATA)>
                                      <!ELEMENT Y (#PCDATA)>]>         <!ELEMENT T (#PCDATA)>
                                                                       <!ELEMENT Y (#PCDATA)>
                                                                       <!ELEMENT C (#PCDATA)>]>
```

Fig. 2. Context-free structure of S_1, S_2 and S_3 defined by DTDs

3 Schema mappings of relational and XML data

In data exchange [11], we meet the problem of computing target instances from source instances: let S be a source schema, T a target schema and $\mathcal{M}$ a *schema mapping*, i.e. a formula expressing a relationship between S and T. For a given instance I of S, find an instance J of T such that $\langle I, J\rangle \models \mathcal{M}$. Such an instance J is called a *solution* for I under $\mathcal{M}$.

In the relational data exchange setting, *source-to-target dependencies* (STDs) [1] are usually used to express schema mappings [12, 19]. The STD is a first-order formula:

$$\forall \mathbf{x}(\Phi(\mathbf{x}) \Rightarrow \exists \mathbf{y}\Psi(\mathbf{x}, \mathbf{y})),$$

that after skolemization has the form of the following second-order STD (SO STD):

$$\exists \mathbf{f}\forall \mathbf{x}(\Phi(\mathbf{x}) \wedge \chi(\mathbf{x}, \mathbf{y}) \Rightarrow \Psi(\mathbf{x}, \mathbf{y})), \tag{4}$$

l	$ConFree$	$KeyDep$	$KeyRefDep$	$ValDep$
$S3$	$A{*}P{*}$			
A	$NR{*}$	$(/S3,(A,(N)))$	$(/S3,A,(R))$ **ref** $KeyDep(P)$	
P	$KTY?C?$	$(/S3,(P,(K)))$		$(/S3/P,Y)=f_y(/S3/P/T)$ $(/S3/P,C)=f_c(/S3/P/T)$
N	ϵ	$(/S3/A,(N,()))$		
...	...	...	...	...

Fig. 3. Schema $S_3 = (S3, ConFree, KeyDep, KeyRefDep, ValDep)$, (labels R, K, T, Y, and C are constrained similarly to that of N)

where: (1) $\mathbf{f}$ is a vector of function symbols, $\mathbf{x}$, $\mathbf{y}$ are vectors of variables; (2) Φ is a conjunction of atoms over source schemas; (3) $\chi(\mathbf{x},\mathbf{y})$ is a conjunction of equalities of the form $t = t'$ where t and t' are terms over $\mathbf{f}$, $\mathbf{x}$ and $\mathbf{y}$; (4) Ψ is a conjunction of atoms over target schema; (5) each variable is safe.

In [11] it was shown that SO STDs are strictly more expressive then STDs and are closed under composition. It means that the composition of two SO STD mappings, $\mathcal{M}_{13} = \mathcal{M}_{12} \circ \mathcal{M}_{23}$, is also a SO STD mapping (this is not true for first-order STD mappings). The semantics of the composition of schema mappings is defined by means of the composition of two binary relations.

For schemas containing nested data (such as XML schemas) extensions of the above mentioned formalism were proposed [22, 29, 13]. In [3], *tree-pattern* formulas [2] are used in STDs instead of the relational atoms. Further on, such SO STDs with tree-pattern formulas will be referred to as SO XSTDs (*second-order XML source-to-target dependencies*). In this way correspondences between XML data can be expressed. However, such mappings do not take key constraints into consideration and they are not capable of ensuring that the target instance conforms to a target schema with key specification. Specification in [3] is restricted to DTD schemas only. To illustrate this problem suppose that we want to restructure the instance I_1 (Fig. 1) of D_1 under D_2 (Fig. 2).

The following XSTD formula specifies a mapping from D_1 to D_2:

$$\begin{aligned}\mathcal{M} := \forall x_T, x_N, x_U(/S1[P[T = x_T \wedge A[N = x_N \wedge U = x_U]]]\\ \Rightarrow \exists x_Y /S2[A[N = x_N \wedge U = x_U \wedge P[T = x_T \wedge Y = x_Y]]]).\end{aligned}$$

Using the specification, two possible target instances, I_2 and I_2' (Fig. 1) can be produced. These instances conform to D_2. However, I_2 and I_2' conforms to difficult sets of keys – given in (2) and (3), respectively. In [3], a *repair* process is also proposed which allows for inventing null values in the final XML tree and to state that some null values should be equal.

In the next sections we will show how both key constraints and value dependencies can be captured by mappings.

4 Key-preserving XML schema mappings

In this paper we propose *key-pattern* formulas, which are a kind of tree-pattern formulas intended to reflect keys in the target schema. A key-pattern formula

SD_1:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="S1">
    <complexType>
      <sequence>
        <element ref="P" minOccurs="0"
     maxOccurs="unbounded" />
      </sequence>
    </complexType>
  </element>
  <element name="P">
    <complexType>
      <sequence>
        <element name="T" type="string" />
        <element ref="A" minOccurs="1"
                 maxOccurs="unbounded" />
      </sequence>
    </complexType>
    <key name="PKey">
      <selector xpath="." />
      <field xpath="T" />
    </key>
  </element>
  <element name="A">
    <complexType>
      <sequence>
        <element name="N" type="string" />
        <element name="U" type="string"
                 minOccurs="0" />
      </sequence>
    </complexType>
    <key name="AKey">
      <selector xpath="." />
      <field xpath="N" />
    </key>
    <valdep>
      <target name="U" />
      <function name="fu" />
      <source xpath="N" />
    </valdep>
  </element>
</schema>
```

SD_2:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="S2">
    <complexType>
      <sequence>
        <element ref="A" ... />
      </sequence>
    </complexType>
  </element>
  <element name="A">
    ...
  </element>
  ...
</schema>
```

SD_3:

```
<schema xmlns="www.w3.org/2001/XMLSchema">
  <element name="S3">
    <complexType>
      <sequence>
        <element ref="A" ..." />
        <element ref="P" ..." />
      </sequence>
    </complexType>
  </element>
  <element name="A">
    ...
    <keyref name="AKeyref" refer="PKey">
      <selector xpath="." />
      <field xpath="R" />
    </keyref>
  </element>
  <element name="P">
    ...
    <key name="PKey">
      <selector xpath="." />
      <field xpath="K" />
    </key>
    ...
  </element>
</schema>
```

Fig. 4. XSD specification for S_1, S_2 and S_3; keys, key references and value dependencies are defined by means of `<key>`, `<keyref>`, and `<valdep>` elements, respectively

is used in a chasing algorithm to control computation of the expected solution. Tree-patterns are in fact predicates of XPath 2.0 [26] with variables in patterns ranging over possible text values. Our definition extends tree-pattern formulas discussed in [3].

If L is a set of labels, then a tree-pattern formula is defined as follows:

Definition 5. *A tree-pattern formula π over L is an expression conforming to the following syntax (P is a path):*

$$\pi ::= P[E] \mid \pi/P[E]$$
$$E ::= x \mid C$$
$$C ::= P = x \mid \pi \mid C \wedge C$$ □

A tree-pattern formula is evaluated in a node of an XML tree I under a valuation ω, where ω is a total function from the set of all variables occurring in π to Str. We denote by $\omega(x)$ the value of ω on x (we assume $\omega(x) = \bot$, if there is not any text value for x in I – this is possible because of semistructured nature of XML).

Further on, by $[\![\pi]\!]$ and $n[\![\pi]\!]$ we will denote a value of π according to semantics of XPath expressions [25, 14], i.e. a set of nodes reachable from the root or from the context node n, respectively, via π.

Definition 6. *The satisfaction of a tree-pattern formula under a valuation ω in a node n of an XML tree I, is defined as follows ($val(n)$ is the text value of the text node n):*

1. $(I, n) \models (x)(\omega)$ *iff* $val(n) = \omega(x)$.
2. $(I, n) \models (P = x)(\omega)$ *iff there is a node* $n' \in n[\![P]\!]$ *such that* $(I, n') \models (x)(\omega)$.
3. $(I, n) \models (C_1 \wedge C_2)(\omega)$ *iff* $(I, n) \models (C_1)(\omega)$ *and* $(I, n) \models (C_2)(\omega)$.
4. $(I, n) \models (P[E])(\omega)$ *iff there is a node* $n' \in n[\![P]\!]$ *such that* $(I, n') \models (E)(\omega)$.
5. $(I, n) \models (\pi/P[E])(\omega)$ *iff* $(I, n) \models (\pi)(\omega)$ *and there is a node* $n' \in n[\![\pi/P]\!]$ *such that* $(I, n') \models (E)(\omega)$.

We say that a formula ϕ is satisfied in (I, n), or that (I, n) satisfies ϕ, denoted $(I, n) \models \phi$, iff $(I, n) \models \phi(\omega)$ for some valuation ω over ϕ into I. $I \models \phi$ denotes that ϕ is satisfied in the root of I. □

Definition 7. *A key-pattern formula δ is a tree-pattern formula restricted to the syntax:*

$\delta ::= /top/l[E] \mid \delta/l[E]$
$E ::= x \mid C$
$C ::= P = x \mid C \wedge C$ □

Lemma 1. *Let $\delta/l[E]$ be a key-pattern formula, and I be an XML tree. Then:*

$$I \models \delta/l[E] \Leftrightarrow I \models \delta \wedge \exists n \in [\![\delta]\!]((I, n) \models l[E]). \tag{5}$$

Proof. The lemma follows from (4) and (5) in Definition 6.

The above lemma will be used in Section 5 as the basis of an algorithm creating key-preserving target instances. Key-pattern formulas can be automatically generated from keys defined in a schema $S = (top, ConFree, KeyDep,$
$KeyRefDep, ValDep)$ over a set
L of labels, by means of the following recursive algorithm:

Algorithm 1 *key-patt(key)*, generation of a key-pattern formula representing the given key.

```
Input : Schema $S = (top, ConFree, KeyDep, KeyRefDep, ValDep)$
                    over a set $L$ of labels,
        label $l \in L - \{top\}$,
        key $\kappa = KeyDep(l) = (P, (l, (P_1, ..., P_k)))$, $k \geq 0$.
Output : Key-pattern formula key-patt($\kappa$).
key-patt($\kappa$) = case of ($\kappa$)
   $(/top, (l, ()))$            :  $/top/l[x]$
```

```
    $(/top, (l, (P_1, ..., P_k)))$ : $/top/l[P_1 = x_1 \wedge \cdots \wedge P_k = x_k]$
    $(P/l', (l, ()))$ : $key\text{-}patt(KeyDep(l'))/l[x]$
    $(P/l', (l, (P_1, ..., P_k)))$ : $key\text{-}patt(KeyDep(l'))/l[P_1 = x_1 \wedge \cdots \wedge P_k = x_k]$
endcase
```
□

For keys of S_1 specified in (1), Algorithm 1 generates key-pattern formulas presented in Table 1.

Table 1. Key-pattern formulas generated by Algorithm 1 for S_1

l	$KeyDep(l)$	$key\text{-}patt(KeyDep(l))$
P	$(S1, (P, (T)))$	$/S1/P[T = x_T]$
T	$(S1/P, (T, ()))$	$/S1/P[T = x_T]/T[x_T]$
A	$(S1/P, (A, (N)))$	$/S1/P[T = x_T]/A[N = x_N]$
N	$(S1/P/A, (N, ()))$	$/S1/P[T = x_T]/A[N = x_N]/N[x_N]$
U	$(S1/P/A, (U, ()))$	$/S1/P[T = x_T]/A[N = x_N]/U[x_U]$

Among schema mappings, we will distinguish:

- *automappings* – a special kind of mappings from a schema onto itself (a special kind of t-mappings),
- *c-mappings* – mappings defining the correspondences between schemas;
- *t-mappings* – mappings defining the key-preserving transformations of schema instances.

4.1 Automappings

The automappings are identity mappings over schemas that describe how instances of the schema are transformed onto themselves. An automapping captures constraints specified within a schema by means of SO XSTD of the form:

$$\pi(\mathbf{x}) \wedge \psi(\mathbf{x}) \Rightarrow \Delta(\mathbf{x}), \qquad (6)$$

where: $\pi(\mathbf{x})$ is a tree-pattern formula capturing the structure of the schema; $\psi(\mathbf{x})$ is a conjunction of atoms of the form $x = x'$ and $x = f(x_1, ..., x_n)$, where the former captures a key reference and the latter – a value dependence; $\Delta(\mathbf{x})$ is a conjunction of key-pattern formulas, where every formula represents a key in the schema. We assume that function symbols are quantified existentially, while variables – universally.

Example 2. Automappings of S_1, S_2 and S_3, referring to XSDs in Fig. 4, have the following specifications:

1. $\mathcal{A}_1 := \pi_1(x_T, x_N, x_U) \wedge \psi_1(x_T, x_N, x_U) \Rightarrow \Delta_1(x_T, x_N, x_U)$, where
   - $\pi_1 := /S1[P[T = x_T \wedge A[N = x_N \wedge U = x_U]]]$

  - $\psi_1 := x_U = f_U(x_N)$
  - $\Delta_1 := /S1/P[T = x_T]$
    $\wedge /S1/P[T = x_T]/T[x_T]$
    $\wedge /S1/P[T = x_T]/A[N = x_N]$
    $\wedge /S1/P[T = x_T]/A[N = x_N]/N[x_N]$
    $\wedge /S1/P[T = x_T]/A[N = x_N]/U[x_U]$
2. $\mathcal{A}_2 := \pi_2(y_T, y_N, y_U, y_Y) \wedge \psi_2(y_T, y_N, y_U, y_Y) \Rightarrow \Delta_2(y_T, y_N, y_U, y_Y)$, where
  - $\pi_2 := /S2[A[N = y_N \wedge U = y_U \wedge P[T = y_T \wedge Y = y_Y]]]$
  - $\psi_2 := y_U = f_U(y_N) \wedge y_Y = f_Y(y_T)$
  - $\Delta_2 := /S2/A[N = y_N]$
    $\wedge /S2/A[N = y_N]/N[y_N]$
    $\wedge /S2/A[N = y_N]/U[y_U]$
    $\wedge /S2/A[N = y_N]/P[T = y_T]$
    $\wedge /S2/A[N = y_N]/P[T = y_T]/T[y_T]$
    $\wedge /S2/A[N = y_N]/P[T = y_T]/Y[y_Y]$
3. $\mathcal{A}_3 :=$
   $/S3[A[N = z_N \wedge R = z_R] \wedge P[K = z_K \wedge T = z_T \wedge Y = z_Y \wedge C = z_C]]$
   $\wedge z_R = z_K \wedge z_Y = f_Y(z_T) \wedge z_C = f_C(z_T) \Rightarrow /S3/A[N = z_N]\wedge$
   $/S3/A[N = z_N]/N[z_N] \wedge /S3/A[N = z_N]/R[z_R] \wedge /S3/P[K = z_K]\wedge$
   $/S3/P[K = z_K]/T[z_T] \wedge /S3/P[K = z_K]/Y[z_Y] \wedge /S3/P[K = z_K]/C[z_C]$

4.2 C-mappings

A *c-mapping* specifies the correspondence between source and target schemas, and states how patterns in the source tree correspond to patterns in the target tree. They are SO XSTDs of the form:

$$\pi_S(\mathbf{x}) \wedge \phi_{S,T}(\mathbf{x}, \mathbf{y}) \Rightarrow \pi_T(\mathbf{x}, \mathbf{y}), \qquad (7)$$

where: $\mathbf{x}$ and $\mathbf{y}$ are source and target variables, respectively; $\pi_S(\mathbf{x})$ is a tree-pattern formula over a source schema S and defines source variables; $\phi_{S,T}(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms of the form $x = x'$, $y = y'$ and $y = f(x_1, ..., x_n)$, where $x = x'$ and $y = y'$ are restrictions over variables, and $y = f(x_1, ..., x_n)$ defines a target variable by means of the source ones; $\pi_T(\mathbf{x}, \mathbf{y})$ is a tree-pattern formula over a target schema T.

Discovering c-mappings or *correspondences* between schemas is a crucial problem in schema mappings and may involve variety of methods and tools [23, 7, 8]. However, formal methods for representing mappings based on well established formal languages enable inferring new mappings in a repository of accepted mappings [17, 11, 19, 9, 20]. In this way *user attention* [15] can be utilized.

Example 3. A c-mapping from S_1 into S_2 has the following specification:

$$\mathcal{M}_{12} := \pi_1(z_T, z_N, z_U) \wedge \phi_{12}(z_T, z_N, z_U, z_Y) \Rightarrow \pi_2(z_T, z_N, z_U, z_Y),$$

where $\pi_1(z_T, z_N, z_U)$ and $\pi_2(z_T, z_N, z_U, z_Y)$ are tree-pattern formulas specified in automappings of S_1 and S_2, respectively, and $\phi_{12}(z_T, z_Y)$ is the atom formula $z_Y = f_Y(z_T)$ defining target variable z_Y. $\square$

4.3 T-mappings

A *t-mapping* or a *transformation* is a specification that describes how data structured under the source schema is to be transformed into data structured under the target schema preserving keys in the target. A t-mapping $\mathcal{T}_{ST}$ from a source schema S into a target schema T is a SO XSTD formula of the form

$$\pi_S(\mathbf{x}) \wedge \psi_{S,T}(\mathbf{x}, \mathbf{y}) \Rightarrow \Delta_T(\mathbf{x}, \mathbf{y}), \tag{8}$$

and can be automatically derived from the c-mapping $\mathcal{M}_{ST}$ from S to T and the automappings $\mathcal{A}_T$ over T. Then the following rule is used:

$$\frac{\begin{array}{c}\mathcal{M}_{ST} = \pi_S(\mathbf{x_1}) \wedge \phi_{ST}(\mathbf{x_1}, \mathbf{y_1}) \Rightarrow \pi_T(\mathbf{x_1}, \mathbf{y_1}) \\ \mathcal{A}_T = \pi_T(\mathbf{x_2}) \wedge \psi_T(\mathbf{x_2}, \mathbf{y_2}) \Rightarrow \Delta_T(\mathbf{x_2}, \mathbf{y_2})\end{array}}{\mathcal{T}_{ST} = (\pi_S(\mathbf{x_1}) \wedge \phi_{ST}(\mathbf{x_1}, \mathbf{y_1}))[(\mathbf{x_1}, \mathbf{y_1}) \mapsto \mathbf{x_2}] \wedge \psi_T(\mathbf{x_2}, \mathbf{y_2}) \Rightarrow \Delta_T(\mathbf{x_2}, \mathbf{y_2})}$$

The formula $(\pi_S(\mathbf{x_1}) \wedge \phi_{ST}(\mathbf{x_1}, \mathbf{y_1}))[(\mathbf{x_1}, \mathbf{y_1}) \mapsto \mathbf{x_2}]$ arises from $\pi_S(\mathbf{x_1}) \wedge \phi_{ST}(\mathbf{x_1}, \mathbf{y_1})$ by replacing variables in $(\mathbf{x_1}, \mathbf{y_1})$ with corresponding variables in $\mathbf{x_2}$. The replacement is made according to occurrences of variables within $\pi_T(\mathbf{x_1}, \mathbf{y_1})$ and $\pi_T(\mathbf{x_2})$.

Example 4. From the c-mapping $\mathcal{M}_{12}$ (Example 3) and the automapping $\mathcal{A}_2$ (Example 2), the following t-mapping $\mathcal{T}_{12}$ from S_1 into S_2 can be obtained:

$$\begin{aligned}\mathcal{T}_{12} := \pi_1(y_T, y_N, y_U) \wedge \phi_{12}(y_T, y_N, y_U, y_Y) \wedge \psi_2(y_T, y_N, y_U, y_Y) \\ \Rightarrow \Delta_2(y_T, y_N, y_U, y_Y)\end{aligned}$$

5 Creation of key-preserving target instances

In this section we propose an algorithm generating a target instance for a given t-mapping. The left-hand side of the t-mapping provides a set Ω of variable valuations, and the right-hand side consists of a set of key-pattern formulas. Each key-pattern formula is used to create a Skolem term that for valuations in Ω delivers nodes of the expected target XML tree. We will show that the created instance satisfies keys represented by key-pattern formulas.

Let δ be a key-pattern formula with variables in $\mathbf{x}$. If for each valuation ω over δ into I the set $[\![\delta(\omega)]\!]$ has exactly one element, then a key represented by δ is satisfied in I. If δ' is satisfied in I and δ is a key-pattern formula of the form $\delta'/l[E]$ then, using Lemma 1, we can recursively analyze whether δ represents a key satisfied in I or not.

The following theorem formulates a necessary condition for satisfaction of a key in an XML tree I. We will show that if the condition is satisfied in I by a key-pattern formula δ, then the key represented by δ is satisfied in I. That observation will be the base of our implementation (Algorithm 2).

Theorem 1. *Let $\delta = \delta'/l[E]$ be a key-pattern formula and I be an XML tree:*

– $\delta' = /top/l_1[E_1]/.../l_m[E_m]$;
– $E = (P_1 = x_1 \wedge \cdots \wedge P_k = x_k)$;
– Ω – *a set of all variable valuations over* δ *into text values in* I.

If for any valuation $\omega \in \Omega$

$$count([\![\delta'/l(\omega)]\!]) = count\{\omega \in \Omega \mid I \models \delta(\omega)\} \tag{9}$$

then the key represented by δ *is satisfied in* I*, i.e.*

$$I \models (/top/l_1/.../l_m, (l, (P_1, ..., P_k))).$$

Proof. (*Sketch*) The key represented by δ' is satisfied in I if for each valuation $\omega \in \Omega$, a set $[\![\delta'(\omega)]\!]$ has exactly one element. By structural induction we will proof that for any $\omega \in \Omega$ the equality (9) implies $count([\![\delta(\omega)]\!]) = 1$. Indeed:

1. The thesis holds for $\delta = /top/l[P_1 = x_1 \wedge \cdots \wedge P_k = x_k]$, since:
   – $count([\![/top(\omega)]\!]) = 1$, for any $\omega \in \Omega$,
   – if the number of nodes of type l determined by $/top/l$ is equal to the number of different valuations of variables $(x_1, ..., x_k)$, then the key $(/top, (l, (P_1, ..., P_k)))$ represented by δ is satisfied in I.
2. Let $\delta = \delta'/l[P_1 = x_1 \wedge \cdots \wedge P_k = x_k]$. Then from the induction hypothesis $count([\![\delta'(\omega)]\!]) = 1$, for each $\omega \in \Omega$. Now it is easy to see that the equality (9) implies that $count([\![\delta(\omega)]\!]) = 1$, for each $\omega \in \Omega$; hence the key represented by δ is satisfied in I. □

The theorem states that if the number of nodes in $[\![\delta'/l]\!]$ is equal to the number of different valuations of variables in δ satisfying I, then the key represented by δ is satisfied in I.

This observation is the basis of our algorithm for generation of key-preserving instances in data exchange. The one-to-one (bijective) correspondence between sets mentioned in the thesis of the theorem, equality (9), can be achieved by means of node-valued Skolem functions. The list of arguments of a Skolem function F_l consists of all variables $\mathbf{x}$ occurring in δ=*key-patt*(*KeyDep*(l)). Then every valuation $\omega \in \Omega$ creates a unique element node $F_l(\omega(\mathbf{x}))$ of type l. Similarly for text nodes.

Now, we shall define the semantics for t-mappings. The left-hand side:

$$\pi_S(\mathbf{x}) \wedge \psi_{S,T}(\mathbf{x}, \mathbf{y})$$

of a t-mapping (8), determines a totally ordered set Ω of variable valuations.

The ordering of valuations in Ω is induced by the lexicographic ordering of tuples of text nodes whose values are assigned to the vector $\mathbf{x}$ of variables. By $ord(\omega)$ we denote the ordering position of ω in Ω. To order nodes in a result target tree, we will use the Dewey order encoding [24], where each node n is assigned a sequence $Pos(n)$ that represents position of n in the tree. For example, $Pos(n) = 1.3.2$ says that n is the second child of the third child of the top element.

We assume:

– $Pos(r) = 0$, if r is the root; $Pos(n) = 1$, if n is the top element;
– $Pos(n) = Pos(n').ord(\omega)$, where n' is the parent of n and n' is obtained under a valuation $\omega \in \Omega$.

Let Ω be a set of valuations, and $\Delta = \{\delta_1, ..., \delta_p\}$ be a set of key-pattern formulas occurring in the right-hand side of a t-mapping (8).

Then an XML tree $J = semT(\Delta)(\Omega)$, where $J = (r, N^e, N^t, \leq, child, \lambda, \nu, \sigma)$, is created as follows:

$$semT(\Delta)(\Omega) = \{semT(\delta)(\omega) \mid \delta \in \Delta, \omega \in \Omega\},$$

where $semT(\delta)(\omega)$ is computed by means of the following recursive algorithm:

Algorithm 2 $semT(\delta)(\omega)$ – generation of XML tree fragment corresponding to a key-pattern formula δ under valuation of ω.

```
Input :  $\delta = /top/l_1[E_1](\mathbf{x_1})/.../l_m[E_m](\mathbf{x_m})$ – a key-pattern formula,
         $\&doc$ – identifier of the result tree.
Output : A fragment of XML tree.
$semT(\delta)(\omega) =$ case of $(\delta)$
  $/top/l_1$ :
    begin
      $r = newNode(\&doc);\ \sigma(r) = \&doc;\ Pos(r) = 0;$
      $n = newNode(top);\ \lambda(n) = l_1;\ Pos(n) = 1;$
      $N^e = N^e \cup \{n\};\ child := child \cup \{(r, n)\};$
    end;
  $\delta'/l_{i-1}[E_{i-1}](\mathbf{x_{i-1}})/l_i[E_i](\mathbf{x_i})$ :
    begin
      $n = F_{l_{i-1}}(\omega(\mathbf{x_{i-1}}));\ n' = F_{l_i}(\omega(\mathbf{x_i}));$
      $\lambda(n') = l_i;\ Pos(n') = Pos(n).ord(\omega);$
      $N^e = N^e \cup \{n'\};\ child := child \cup \{(n, n')\};$
      $semT(\delta'/l_{i-1}[E_{i-1}](\mathbf{x_{i-1}}))(\omega);$
    end;
  $\delta'/l_{i-1}[E_{i-1}](\mathbf{x_{i-1}})/l_i[x](\mathbf{x_i})$ :
    begin
      $n = F_{l_{i-1}}(\omega(\mathbf{x_{i-1}}));\ n' = F_{l_i}(\omega(\mathbf{x_i}));\ n'' = newNode();$
      $\lambda(n') = l_i;\ Pos(n') = Pos(n).ord(\omega);$
      $N^e = N^e \cup \{n'\};\ child := child \cup \{(n, n')\};$
      $\nu(n'') = \omega(x);\ Pos(n'') = Pos(n').ord(\omega);$
      $N^t = N^t \cup \{n''\};\ child := child \cup \{(n', n'')\};$
      $semT(\delta'/l_{i-1}[E_{i-1}](\mathbf{x_{i-1}}))(\omega);$
    end;
endcase
```

According to the proposed semantics, one can show that for mappings in Example 1 we have $\mathcal{T}_{12}(I) = I_2$ (see Fig. 1). Some null values in I_2 are equal in force of the value dependency $x_Y = f_Y(x_T)$ specified in ϕ_{12}. Value dependencies specified within mappings may be used to infer some missing data. It is especially useful when we integrate data from many different sources, for example using a method proposed in [20].

6 Conclusions

We proposed a new approach to use XML schema mappings in data exchange. To this aim we introduced a new class of tree-pattern formulas called *key-pattern* formulas that provide a natural way to express contextual constraints in XML schema mappings and are used to perform key-preserving mappings. We distinguished among three classes of mappings: automappings (*transformations of a schema onto itself*), c-mappings (*correspondences*) and t-mappings (*key-preserving transformations*). The distinction is justified by different ways in which they are obtained: automatically from a schema specification (automappings), quasi-manually (c-mappings), or by deriving from another mappings (t-mappings). In our formalism, mappings capture three kinds of contextual constraints: keys, key references and value dependencies. We demonstrated benefits of this formalism including increased specification accuracy, and the ability to manage mappings and reasoning on them. In the future we plan to consider operation on key-preserving mappings, as well as on mappings capturing another constraints. For this purpose we need reasoning procedures on tree-patterns and ontological knowledge describing the semantics of XML schemas which may be applied for finding correspondences between different schemas [6].

References

1. Abiteboul, S., Hull, R., Vianu, V.: *Foundations of Databases*, Addison-Wesley, Reading, Massachusetts, 1995.
2. Amer-Yahia, S., Cho, S., Lakshmanan, L. V. S., Srivastava, D.: Tree pattern query minimization., *VLDB Journal*, **11**(4), 2002, 315–331.
3. Arenas, M., Libkin, L.: XML Data Exchange: Consistency and Query Answering, *PODS Conference*, 2005, 13–24.
4. Buneman, P., Davidson, S. B., Fan, W., Hara, C. S., Tan, W. C.: Reasoning about keys for XML, *Information Systems*, **28**(8), 2003, 1037–1063.
5. Calvanese, D., Giacomo, G. D., Lenzerini, M., Rosati, R.: Logical Foundations of Peer-To-Peer Data Integration., *Proc. of the 23rd ACM SIGMOD Symposium on Principles of Database Systems (PODS 2004)*, 2004, 241–251.
6. Cybulka, J., Meissner, A., Pankowski, T.: Schema- and Ontology-Based XML Data Exchange in Semantic E-Business Applications, *Business Information Systems, BIS 2006, Lecture Notes in Informatics, Vol.85*, 2006, 429–441.
7. Doan, A., Domingos, P., Halevy, A.: Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach, *ACM SIGMOD 2001*, ACM, 2001, 509–520.
8. Doan, A., Noy, N. F., Halevy, A. Y.: Introduction to the Special Issue on Semantic Integration., *SIGMOD Record*, **33**(4), 2004, 11–13.
9. Fagin, R.: Inverting schema mappings., *PODS Conference*, 2006, 50–59.
10. Fagin, R., Kolaitis, P. G., Miller, R. J., Popa, L.: Data Exchange: Semantics and Query Answering., *ICDT 2003*, Lecture Notes in Computer Science 2572, Springer, 2002, 207–224.
11. Fagin, R., Kolaitis, P. G., Popa, L.: Data exchange: getting to the core., *ACM Trans. Database Syst.*, **30**(1), 2005, 174–210.

12. Fagin, R., Kolaitis, P. G., Popa, L., Tan, W. C.: Composing schema mappings: Second-order dependencies to the rescue., *ACM Trans. Database Syst.*, **30**(4), 2005, 994–1055.
13. Fuxman, A., Hernández, M. A., Ho, C. T. H., Miller, R. J., Papotti, P., Popa, L.: Nested Mappings: Schema Mapping Reloaded., *VLDB*, 2006, 67–78.
14. Gottlob, G., Koch, C., Pichler, R.: Efficient Algorithms for Processing XPath Queries, *Proc. of the 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, China*, 2002, 95–106.
15. Halevy, A. Y., Rajaraman, A., Ordille, J. J.: Data Integration: The Teenage Years., *VLDB*, 2006, 9–16.
16. Lenzerini, M.: Data Integration: A Theoretical Perspective., *PODS*, 2002, 233–246.
17. Madhavan, J., Halevy, A. Y.: Composing Mappings Among Data Sources., *VLDB*, 2003, 572–583.
18. Melnik, S., Bernstein, P. A., Halevy, A. Y., Rahm, E.: Supporting Executable Mappings in Model Management., *SIGMOD Conference*, 2005, 167–178.
19. Nash, A., Bernstein, P. A., Melnik, S.: Composition of Mappings Given by Embedded Dependencies., *PODS*, 2005.
20. Pankowski, T.: Management of executable schema mappings for XML data exchange, *Database Technologies for Handling XML Information on the Web, EDBT 2006 Workshops*, Lecture Notes in Computer Science **4254**, Springer, 2006, 264–277.
21. Pankowski, T., Hunt, E.: Data Merging in Life Science Data Integration Systems, *Intelligent Information Systems, New Trends in Intelligent Information Processing and Web Mining*, Advances in Soft Computing, Springer Verlag, 2005, 279–288.
22. Popa, L., Velegrakis, Y., Miller, R. J., Hernández, M. A., Fagin, R.: Translating Web Data., *VLDB*, 2002, 598–609.
23. Rahm, E., Bernstein, P. A.: A survey of approaches to automatic schema matching, *The VLDB Journal*, **10**(4), 2001, 334–350.
24. Tatarinov, I., Viglas, S., Beyer, K. S., Shanmugasundaram, J., Shekita, E. J., Zhang, C.: Storing and querying ordered XML using a relational database system., *SIGMOD Conference*, 2002, 204–215.
25. Wadler, P.: Two Semantics for XPath, Available on: http://homepages.inf.ed.ac.uk/wadler/papers/xpath-semantics/xpath-semantics.pdf, 2000. 26 July 1999, revised 4 January 2000
26. XML Path Language (XPath) 2.0: 2006. www.w3.org/TR/xpath20
27. XML Schema Part 1: Structures 2d Edition: 2004. www.w3.org/TR/xmlschema-1
28. XQuery 1.0 and XPath 2.0 Data Model.: 2002. www.w3.org/TR/query-datamodel
29. Yu, C., Popa, L.: Constraint-Based XML Query Rewriting For Data Integration., *SIGMOD Conference*, 2004, 371–382.