Appendix to the Paper: Schema Mapping and Query Translation in Heterogeneous P2P XML Databases

Angela Bonifati · Elaine Chang · Terence Ho · Laks V.S. Lakshmanan · Rachel Pottinger · and Yongik Chung

Received: September 18, 2008 / Accepted: July 9, 2009

A Class of Transformations

We have presented a technique that takes as input a set of box and arrow correspondences between a pair of DTDs and infers TreeLog rules involving tree expressions. How do we know that the mappings captured using the inferred rules are meaningful? What can we say about the class of mappings (transformations) that are captured by the rules? To answer these questions, in this appendix we introduce a small set of simple but powerful algebraic operators for expressing transformations between (database instances of) DTDs. We demonstrate that transformations captured by expressions in this algebra are expressible as a set of TreeLog rules.

The operators consist of three main groups: (1) unnest and nest, (2) flip and flop, and (3) split and merge. Additionally, we also allow individual node insertion, deletion, and tag modification for “completeness”. These latter operations are needed to introduce node types in one schema that may have no counterpart in the other.

Unnest/Nest Example: The first group of operators converts a nested representation into a flattened or unnested one and vice versa. For example, Figure 1 shows two DTDs. The DTD in Figure 1(b) is obtained by unnest the element type Patient in Figure 1(a) on its child sub-element Symptom. The associations between a patient and his/her symptoms are captured by a reference link from symptom to patient in Figure 1(b) via @PatientRef → @Id. Conversely, nesting Symptom into Patient in the DTD of Figure 1(b) yields the DTD of Figure 1(a). Note that the transformation sketched in Figure 1 at the schema level induces a corresponding obvious transformation on the instances.

We formalize these operators below.

Definition 1 [Unnest] Consider a DTD ∆ and two element type nodes A, B such that ∆ contains the edge (A, B) with label ℓ ∈ {‘∗’, ‘+’}. Suppose A has an ID attribute @id. Let D be a valid database instance of ∆. Then the unnest operator UA,B(D) is an instance of the following DTD ∆′, obtained from ∆: B is made a child of the root instead of its current parent type A and the edge (root, B) is labeled ℓ; B has an additional IDREF attribute, @ARef, which points to the ID attribute @id of A. The instance ∆′ = UA,B(D) is obtained from D as follows: (i) make every element instance b of B that is a child of an element instance a of A, an immediate child of the root; (ii) set the value of the @ARef attribute of b to match the value of attribute @id of a.

Note that A need not be a child of the root in Definition 1. We could also define unnest in such a way that B becomes a sibling of A, a variant that we omit. Also, unnest is well defined on arbitrary DTDs. We chose to give a simple version of this operator for ease of exposition. Finally, even if A in the input DTD does not have an ID attribute, it can always be added in the transformed DTD and instance.

We demonstrate that the Unnest transformation can be expressed in TreeLog. E.g., the Unnest operation that transforms instances of the DTD of Figure 1(a) to Figure 1(b) can be expressed as the rules:

\[
\begin{align*}
1. & \quad \text{Records} \rightarrow f(\text{Patient})[\text{Problem} \rightarrow \text{f}](\text{Id} \rightarrow \text{f}1, \text{Name} \rightarrow \text{f}N]) \\
2. & \quad \text{Records} \rightarrow f(\text{Patient})[\text{Symptoms} \rightarrow \text{g}](\text{Id} \rightarrow \text{f}1, \text{Problem} \rightarrow \text{f}P) \\
& \quad \quad \text{Date} \rightarrow \text{g}D, \text{PatientRef} \rightarrow \text{f}1] \\
3. & \quad \text{Records} \rightarrow f(\text{Patient})[\text{Symptoms} \rightarrow \text{g}](\text{Id} \rightarrow \text{f}1, \text{Symptoms} \rightarrow \text{g}S[\text{Problem} \rightarrow \text{f}P, \\
& \quad \quad \text{Date} \rightarrow \text{g}D]).
\end{align*}
\]

Definition 2 [Nest] Consider a DTD ∆ and two element nodes A, B such that A contains an ID attribute, say @id and B contains an IDREF attribute, say @ARef. Let D be a valid database instance of ∆. Then NA,B(D) is an instance of the following DTD ∆′, obtained from ∆: B is made a child of the element type A and the @ARef IDREF attribute B is deleted. The instance ∆′ = NA,B(D) is obtained from D as follows: (i) make every B element b a child of the A element a such that the @ARef attribute of the former matches the @id attribute of the latter; (ii) delete the @ARef attribute of b.

A. Bonifati (Corresponding Author)
Icar-CNR, Via P. Bucci 41/C
87036 Rende (CS), Italy, Ph:+39-0984-831727, Fax:+39-0984-839054
E-mail: bonifati@icar.cnr.it

E. Chang, T. Ho, L. Lakshmanan, R. Pottinger, Y. Chung
UBC, 2366 Main Mall,
Vancouver,B.C.,Canada,V6T 1Z4
E-mail: {echang,terenho,laks,rap,ychung25}@cs.ubc.ca

Fig. 1 Illustrating Unnest/Nest.
As an illustration of expressing the Nest transformation in TreeLog, the Nest operation that transforms instances of the DTD of Figure 1(b) to Figure 1(a) can be expressed as the rules:

1. \( R_1 \rightarrow f(S_R)[\text{Patient} \rightarrow f(S_I)[\text{Id} \rightarrow \$I, \text{Name} \rightarrow \$N]] \)

2. \( R_2 \rightarrow f(S_R)[\text{Symptoms} \rightarrow f(S_S)] \)

\( \text{Problem} \rightarrow f(P, \text{Date} \rightarrow \$D) \)

\( R_3 \rightarrow f(S_R)[\text{Id} \rightarrow \$I, \text{Name} \rightarrow \$N, \text{Ailment} \rightarrow \$A] \)

\( \equiv \Delta \)

Definition 3 [Flip] Consider a DTD \( \Delta \) and element type nodes \( A, B, C \) such that \( \Delta \) contains the edges \((A, B), (B, C)\), and \( C \) is a leaf. Let \( D \) be a valid database instance of \( \Delta \). Then the flip operator \( \Phi_{B,C}(D) \) is an instance of the following DTD \( \Delta' \), obtained from \( \Delta \) over \( \Delta_i \): (i) remove the edges \((A, B)\) and \((B, C)\), and for every value of \( C \) in database \( D \), create a node with tag \( C \) and add the edges \((A, C), (C, B)\); (ii) the labels of the edges \((A, C), (C, B)\) are all identical to the label on the edge \((A, B)\) in \( \Delta \), and the labels of the edges \((C, B)\) are all \'1\'. The instance \( D' = \Phi_{B,C}(D) \) is obtained from \( D \) as follows: for each \( A \) element \( a \): (i) let \([b_1, .., b_m]\) be the list of \( B \) sub-elements of \( a \); delete the edges \((a, b_i)\); (ii) for each \( b_i \), delete the edge \((b_i, c)\), create a new node \( c \) in \( D' \) whose tag is the value of the node \( c \) in \( D \), and make \( c \) a child of \( a \); make a copy of the element \( b_i \), a sub-element child of the node \( c \).

Note that in the database \( D \), if a \( B \) element does not have a \( C \) sub-element (say because the \( (B, C) \) edge was labeled \'0\' in the DTD \( \Delta \)), then this \( B \) element will not be present in the transformed database \( D' \). Similarly, if a \( B \) element has \( k > 1 \) \( C \) sub-elements with values \( c_1, .., c_k \), then a copy of this \( B \) element (without the \( C \) sub-elements) will appear \( k \) times in \( D' \), one under each of the nodes associated with the \( c_i \)'s. In our example in Figure 2, the edge \((\text{Patient}, \text{Ailment})\) is labeled \'1\' so, every patient in an instance of Figure 2(a) will be present exactly once in the transformed instance of Figure 2(b).

Figure 3(a)-(b) illustrates the flip operator using instances of the DTD in Figure 2(a)-(b). When we flip the instance of Figure 3(a) w.r.t. Ailment and Patient, we obtain the instance of Figure 3(b). Applying flop on the instance of Figure 3(b) w.r.t. Ailment and Patient yields the instance of Figure 3(a).

This transformation can be expressed in TreeLog using the following rule:

\( \text{Records} \rightarrow f(S_R)[\text{Name} \rightarrow g(S_A[\text{text()}], \$I)] \)

\( \equiv \Delta \)

\( \text{Patient} \rightarrow f(S_P)[\text{Id} \rightarrow \$I, \text{Name} \rightarrow \$N, \text{Ailment} \rightarrow \$A] \)

\( \text{Symptoms} \rightarrow f(S_S)[\text{Symptom} \rightarrow f(S_S)] \)

Definition 4 [Flop] Let \( \Delta \) be a DTD containing element nodes \( A, B_1, .., B_m \) and edges \((A, b_i), (b_i, C)\), \( 1 \leq i \leq k \). Suppose the label on edges \((A, b_i)\) is \( \ell \). Let \( D \) be a valid database instance of \( \Delta \). Let \( B \) be a new tag that does not appear in \( \Delta \). Then \( \Delta_{B,C}(D) \) is an instance of the DTD \( \Delta' \), obtained from \( \Delta \) as follows: (i) remove all nodes \( b_i \) and replace them with a single node with tag \( B \), make this node a child of \( C \); (ii) make \( C \) a child of \( A \); (iii) the label on the edge \((A, C)\) is \( \ell \). The instance \( D' = \Delta_{B,C}(D) \) is obtained from \( D \) as follows: for each \( C \) element \( c \), with parent \( B \) element \( b \), let \( b \) be the \( A \) element parent of \( b \); (i) then delete \( b \) and make \( c \) a direct child sub-element of \( a \); (ii) create a new child sub-element with tag \( B \) under \( c \) and set its text value to \( b \).

The flop operator is illustrated in Figure 3(a)-(b). Applying \( \Delta_{Ailment, Patient} \) on the instance in Figure 3(b) yields the one in Figure 3(a). Here, the tag \( A \) corresponds to the root tag Records. Note \( A \) is needed for the operator to be well defined but is not included as a parameter in the specification of the flop operator.
Definition 6 [Split] Let $\Delta$ be a DTD containing element nodes $A, b_1, \ldots, b_k, C$ with edges $(A, b_i), (b_i, C)$. Then $S_{A,C}(D)$ is an instance of the DTD $\Delta'$, obtained from $\Delta$ as follows: (i) the label on the edges $(A, b_i)$ is set to $'+'$; (ii) the label on the edges $(b_i, C)$ is set to $'1'$. The instance $S_{A,C}(D)$ is obtained from $D$ as follows: (i) for every $A$ element $a$, for every $B$ sub-element child $b$ of $a$, whenever $b$ has $n > 1$ $C$ sub-element children $c_1, \ldots, c_n$, replace $b$ with $n$ copies of $b_i$; (ii) make $c_i$ the child of the $i$-th copy of $b$.

![Diagram](image)

Fig. 4 Illustrating Merge/Split: (a) DTD; (b) instance.

On our running example, the split operator can be expressed in TreeLog using the following rule:

$$\text{Records} \rightarrow R[S[A \rightarrow g(S[A, S[P] [\text{Patient} \rightarrow S[P])].$$

Expressions over this algebra define database transformations. It is straightforward to show that transformations expressible by composition of the operators defined above can be expressed as sets of rules in TreeLog. E.g., using compositions of the operators defined, we can transform instances of the MON DTD in Figure 2 (see the paper) to those of the BOS DTD and vice-versa. We already saw that these transformations are expressible in TreeLog.