Supplementary Material for ICDE Submission # 458: Querying Heterogeneous P2P XML Databases

Angela Bonifati, Icar CNR, Italy
Elaine Chang, UBC, Canada
Terence Ho, UBC, Canada
Laks V.S. Lakshmanan, UBC, Canada

This supplementary material is being provided for the benefit of the reviewers and may be read at their discretion.

1 Class of Transformations

We have presented a technique for inferring Datalog-like rules involving tree expressions, that express the mapping between a pair of DTDs, based on correspondences specified using boxes and arrows. 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, we introduce a small set of simple but powerful algebraic operators for expressing transformations between (database instances of) DTDs. We will eventually show that the rules inferred by our algorithm based on boxes and arrows correspond to complex expressions obtained by composing these algebraic operators.

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.

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 *unnesting* the element type `Patient` in Figure 1(a) on its child subelement `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 `@Patref` $\rightarrow$ `@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 Δ and two element type nodes A, B such that Δ contains the edge (A, B) with label $\ell \in \{`*', `+'\}$. Suppose A has an ID attribute $@id$. Let D be a valid database instance of Δ. Then the *unnest*

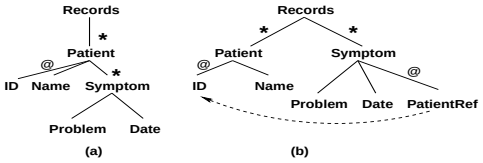

Figure 1. Illustrating Unnest/Nest.

operator $\mathcal{U}_{A:B}(D)$ is an instance of the following DTD Δ', obtained from Δ: 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 $D' = \mathcal{U}_{A:B}(D)$ 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 expositional simplicity. 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.

Definition 2 [Nest] Consider a DTD Δ 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 Δ. Then $\mathcal{N}_{A:B}(D)$ is an instance of the following DTD Δ', obtained from Δ: B is made a child of the element type A and the $ARef$ IDREF attribute B is deleted. The instance $D' = \mathcal{N}_{A:B}(D)$ 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. ■

Flip/Flop Example: Consider Figure 2. In Figure 2(a), there is a list of patients for each of whom their id, name, and ailment are shown. In Figure 2(b), for each patient, the patient is a child of a new node whose tag is the value of

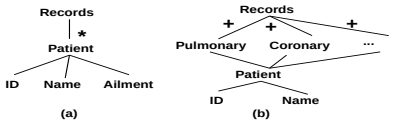

Figure 2. Illustrating Flip/Flop: DTDs.

the patient's ailment (e.g., 'Pulmonary', 'Coronary' etc.). We refer to the operation that transforms instances of Figure 2(a) to those of Figure 2(b) as *flip*. Its converse, *flop*, transforms instances of the DTD Figure 2(b) into corresponding instances of Figure 2(a).

Definition 3 [Flip] Consider a DTD Δ and element type nodes A, B, C such that Δ contains the edges $(A, B), (B, C)$, and C is a leaf. Let D be a valid database instance of Δ. Then the *flip operator* $\mathcal{F}_{B:C}(D)$ is an instance of the following DTD Δ', obtained from Δ: (i) remove the edges (A, B) and (B, C), and for every value c 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) are all identical to the label on the edge (A, B) in Δ, and the labels of the edges (c, B) are all '1'. The instance $D' = \mathcal{F}_{B:C}(D)$ is obtained from D as follows: for each A element a: (i) let $[b_1, ..., b_k]$ be the list of B subelements of a; delete the edges (a, b_i); (ii) for each b_i, for each C subelement c of 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 subelement child of the node c. ∎

Note that in the database D, if a B element does not have a C subelement (say because the (B, C) edge was labeled '?' in the DTD Δ), then this B element will not be present in the transformed database D'. Similarly, if a B element has $k > 1$ C subelements with values $c_1, ..., c_k$, then a copy of this B element (without the C subelements) will appear k times in D', once under each of the nodes associated with the c_i's. In our example in Figure 2, the edge (`Patient`, `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 DTDs 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).

Definition 4 [Flop] Let Δ be a DTD containing element nodes $A, C, b_1, ..., b_k$ and edges $(A, b_i), (b_i, C), 1 \leq i \leq k$. Suppose the label on edges (A, b_i) is ℓ. Let D be a valid database instance of Δ. Let B be a new tag that does not appear in Δ. Then $\mathcal{L}_{B:C}(D)$ is an instance of the DTD Δ', obtained from Δ 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 ℓ. The instance $D' = \mathcal{L}_{B:C}(D)$ is obtained from D as follows: for each C element c, with parent B element b, let a be the A element parent of b; (i) then delete b and make c a direct child subelement of a; (ii) create a new child subelement with tag B under c and set its text value to b. ∎

The flop operator is illustrated in Figure 3(a)-(b). Applying $\mathcal{L}_{\texttt{Ailment:Patient}}$ 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.

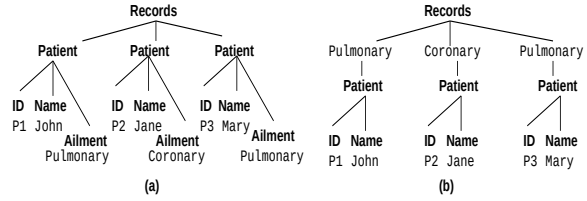

Figure 3. Illustrating Flip/Flop: Instances.

Split/Merge Example: Consider the database in Figure 3(b). We could merge nodes that are siblings and have a common tag, and possibly satisfy some additional conditions. E.g., we could merge all `Pulmonary` nodes and all `Coronary` nodes, thus grouping together all subtrees that were rooted at these nodes in the input instance. The resulting DTD and database instance are shown in Figure 4. Note that merge is associative and is thus well defined. Conversely, splitting `Patient` elements in the instance of Figure 4(b) would yield the instance in Figure 3(b). Conversely, splitting `Patient` elements Figure 4(b) would on the instance of Figure 4(b) would yield the instance in Figure 3(b).

Definition 5 [Merge] Let Δ be a DTD containing element type nodes $A, b_1, ..., b_k, C$ and edges $(A, b_i), (b_i, C)$, $1 \leq i \leq k$. Let D be a valid database instance of Δ. Then the *merge operator* $\mathcal{M}_{A:C}(D)$ is an instance of the DTD Δ', obtained from Δ as follows: (i) the label on the edges (A, b_i) is set to '1'; (ii) the label on the edges (b_i, C) is set to '+'. The instance $D' = \mathcal{M}_{B:C}(D)$ is obtained from D as follows: (i) let $\{b^1, ..., b^m\}$ be the set of all parents of the C elements and are children of a common A element, say a; whenever any two nodes b^i, b^j have the same tag, merge them; (ii) subtrees rooted at b^i, b^j now get rooted at the merged node. ∎

Note that node merging is associative so the operator merge above is well defined. Applying the split operator

$\mathcal{S}_{\texttt{Records:Patient}}(D)$ to the database instance in Figure 4(b) results in the instance of Figure 3(b).

We now turn to split.

Definition 6 [Split] Let Δ be a DTD containing element nodes $A, b_1, ..b_k, C$ with edges $(A, b_i), (b_i, C)$. Then $\mathcal{S}_{A:C}(D)$ is an instance of the DTD Δ', obtained from Δ 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 $\mathcal{S}_{A:C}(D)$ is obtained from D as follows: (i) for every A element a, for every B subelement child b of a, whenever b has $n > 1$ C subelement children $c_1, ..., c_n$, replace b with n copies of b; (ii) make c_i the child of the i-th copy of b. ■

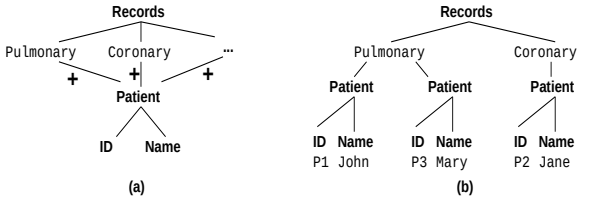

Figure 4. Illustrating Merge/Split: (a) DTD; (b) instance.

Expressions over this algebra define database transformations. E.g., using compositions of the operators defined, we can transform instances of the MonG DTD in Figure 1 in the submitted paper to those of the MasG DTD and vice versa. We omit the details for brevity. We have the following result concerning the class of transformations that are captured by our mapping rules.

Theorem 1 [Rules versus Algebra] : *Let E be any expression obtained by composing the algebraic operators introduced in this section, expressing a transformation from instances of a DTD Δ_1 to those of DTD Δ_2. Then there is a set of mapping rules $\mathcal{R}$ that precisely captures E, i.e., $\forall D_i \in inst(\Delta_1) : \mathcal{R}(D_i) = E(D_i)$. Furthermore, every set of rules inferred by our algorithm expresses a transformation that is captured by some expression over this algebra.* ■

One might ask if we could insert/delete nodes individually, then we can map any database tree to any other. So, what's the relevance of other operations? The problem with such a "transformation" is that it cannot map a database instance to another in a way that preserves associations between data items. This is why the algebraic operators unnest/nest, flip/flop, and merge/split are needed. The role played by node addition/deletion and tag modification is to capture those schema elements which have no counterpart in the other schema.