RA: An XML Schema Reduction Algorithm

Angela C. Duta[1], Ken Barker[1], and Reda Alhajj[1][2]

[1] ADSA Laboratory, Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, Canada
{duta,barker,alhajj}@cpsc.ucalgary.ca
http://www.adsa.cpsc.ucalgary.ca
[2] Department of Computer Science, Global University, Beirut, Lebanon

Abstract. XML file comparison and clustering are two challenging tasks still accomplished predominantly manually. XML schema contains information about data structure, types, and labels found in an XML file. By reducing the XML schema tree to its significant nodes the task of finding structural equivalent schemas, and implicit XML files that refer to the same entities, is simplified.

1 Introduction

1.1 Hypothesis and Methodology

New XML files are added daily to databases triggered by the increased popularity of XML as the new database exchange standard. Organizing XML files in clusters must be based on the information they store related to a set of entities. This task is done by determining some equivalence measure between XML schemas and checking that it is above a minimum threshold. There is much work ([5], [6], [8], [9], [10]) in this area but we believe there is still room to contribute by finding files that refer to the same set of entities if their tree structures are significantly different. To determine schema equivalence for files that have a different organization, we propose a preparatory step for an efficient schema comparison.

The difficulty of comparing and finding matchable schemas arises for two reasons: (1) there are three data storage units in XML: elements, attributes, and text content, and (2) the hierarchical feature of XML structures. Thus, XML schema equivalence must be evaluated from three perspectives: (1) hierarchical structure (structural equivalence), (2) elements and attributes data types (syntactic equivalence), and (3) elements and attributes names (semantic equivalence). This paper focuses on the structural equivalence of XML schema trees. We challenge the current trend in XML equivalence that considers "elements at higher levels ... more relevant than subelements deeply nested" [2] by focusing on leaf nodes. Our argument is that leaf nodes store data in XML data files, while higher level nodes are used primarily to intelligibly organize the information stored in leaves.

1.2 Contribution

By reducing an XML schema to its essential nodes, our method improves the process of determining structural equivalence. This paper defines the essential information that must be extracted from each XML schema so that a comparison between them is efficient. The novelty of our method is to prepare XML schemas for structural matching based on the equivalent leaves content rather than hierarchical context and vicinities. A *leaf content*[3] is defined by (1) data type and (2) number of minimum and maximum occurrences. The Reduction Algorithm (RA) eliminates from XML trees organizational nodes and nodes with no data content; but it preserves the leaf nodes and their connections to other leaf nodes. The result is a simplified structure that stores the same data as the source structure but is efficiently compared to other XML structures. This paper presents the preprocessing step for structural matching that is formed by schema reduction. It will be followed by a full method for determining structural equivalence in the near future.

1.3 Motivating Example

Figures 1 and 2 illustrate two simple examples of DTDs that store data about employees, projects, and tasks for a company. Note that the element data type definitions have not been included. No mechanism has yet appeared in the literature to clearly compare these types of XML schemas to decide if they are equivalent. This paper presents a reduction method for XML schemas to be used before the equivalence algorithm is executed.

1.4 Paper Organization

The balance of this paper is organized as follows. Section 2 briefly discusses several developed methods to evaluate XML schema equivalence. Section 3 defines several concepts used in this paper while Section 4 introduces the Reduction Algorithm which is evaluated in Section 5. This paper draws some conclusions in Section 6.

```
<!ELEMENT company1 (employee+, project*)>
<!ELEMENT employee (eid | sin, name, pid*)>
<!ATTLIST employee address CDATA #REQUIRED>
<!ELEMENT project (pid, description)>
```

Fig. 1. Repeated employee and project elements

[3] Note that a *leaf content* is different from the term *element content* used by W3C [4], which is formed by text and attributes.

```
<!ELEMENT company2 (employee+)>
<!ELEMENT employee (sin, name, address*, projects?)>
<!ATTLIST employee eid CDATA #REQUIRED>
<!ELEMENT projects (project+)>
<!ELEMENT project (description?)>
<!ELEMENT description (#PCDATA)>
<!ATTLIST description budget CDATA #REQUIRED>
```

Fig. 2. Nested structure of employee and project elements

2 RELATED WORK

Database equivalence has been of interest for more than thirty-five years. More recently, XML equivalence is being investigated by the database community. W3C has provided a starting point to address this problem by defining canonical forms for XML [3]. Salminen and Tompa [11] suggest that a better approach should be developed which does not omit any information from the XML source document.

Our work is based on the XML normal form defined by Arenas and Libkin [1]. They define a generalized BCNF for nested XML structures named XNF (XML Normal Form). XNF optimizes two very common problems in XML: update anomalies and redundancy. The algorithm developed to transform any arbitrary DTD into a well-design DTD in XNF is proven to be lossless [1].

A *generic* schema matching algorithm is presented [9] that can be applied to XML, relational, or other schema types. The schema matching is based on automatic linguistic matching (elements' name) and structural matching (schema structure, path matching, constraints, and element data types). Original techniques such as these focus on the leaf nodes and context-dependent matching of shared complex types.

Lee *et al.* [8] propose a method for scalable integration of DTDs. The method is based on two steps (1) finding similar DTDs and (2) generating DTD clusters. The similarity between two DTDs are evaluated from three perspectives: (1) semantic similarity (similarity between node labels, constraints and path context (ascendants)), (2) immediate descendant similarity, and (3) leaf context similarity. Ontological similarity OntologySim [8] (a component of semantic similarity) is based on finding similar labels or synonyms (using WordNet [7]) and the depth (number of similar subsequent characters) for abbreviations (such as emp for employee). Constraints such as +, *, ?, or none are given weights of similarity. The authors have conducted several experiments using more than 150 DTDs to demonstrate their approach has better performance than others. This work is similar to ours in that it addresses some DTD transformation rules also adopted by us. It is different from our work because it evaluates the path context for each element which makes it unsuitable for trees with significant different structures.

The problem of clustering XML documents based on their structural similarity is also addressed by Nierman and Jagadish [10]. A collection of documents with DTD's from the same domain is divided into smaller sets of very similar

DTDs based on the edit distances. The edit distance is calculated using dynamic programming as the minimum cost to transform a tree A into B through operations such as relabel node, insert node, delete node, insert tree, and delete tree. Clusters are formed for DTDs where the edit distance is minimal. This method works for documents with DTDs having the same tree structure but it cannot be applied to trees that have a significant different structure even though they refer to the same domain.

COMA [6] is a system that combines several simple and hybrid matching algorithms. The simple algorithms refer to one of these aspects in DTD: labels, data types, or user input. Hybrid matchers focus either at the element level (i.e. Name and TypeName) or include the structural organization (i.e. NamePath, Children, and Leaves). One novelty of this approach is reusing the previous match results which proved to be efficient in some cases but missed some matchings in other cases. Our approach extends the structural matchers Children and Leaves by combining and generalizing them to any type of node (repeated, optional, alternative options, key, reference, *etc.*)

An evaluation of the most recent approaches in schema matching for relational or XML models is available [5]. Do *et al.* [5] present an objective comparison of different approaches that use different effectiveness evaluation measures. The conclusion of this work is that the effectiveness of these tools is unclear. This is because there has been no uniform system used to evaluate the effectiveness of each tool. Do *et al.* [5] suggest creation of a schema matching benchmark containing real-length XML schemas to be used to test, evaluate, and compare future approaches. We consider including this proposed framework in our next paper that develops a new schema equivalence method based on the Reduction Algorithm described in this paper.

3 ALGORITHM FRAMEWORK

Before presenting the Reduction Algorithm for preprocessing schemas we first define a framework.

Definition 1. *An XML schema is defined as a tuple S = <root, E, A, T> and four mappings type, descendent, minOccurences, and maxOccurences, as follows:*

- *E is a finite set of elements, A a finite set of attributes, and T a finite set of text contents.*
- *root is the only element from which all other elements, attributes or text contents are direct or indirect descendants.*
- *type is a mapping from E, A, or T to the known simple XML data types (integer, string, data etc.). Consider e an element in E, a an attribute in A, and t a text content in T, then the type mappings are defined as follows:*
  - $type(e) = \begin{cases} type(t) : if\ descendent(e) = t \\ empty\ \ : otherwise \end{cases}$
  - *type(t)= integer, string, date, etc.*
  - *type(a)= integer, string, date, etc.*

An attribute or text content cannot be empty. However, an empty element contains subelements and/or attributes but no text content.

- *descendent*[4] *is a mapping from E to itself, A and/or T constructing the hierarchy of an XML schema. Consider e_i an element in E. $E_i \subset E$ is the set of direct subelements of e_i , $A_i \subset A$ the set of e_i's attributes, and t its text content from T. The descendent mappings are defined as follows: descendent(e_i) =*
$$\begin{cases} E_i \cup A_i : \textit{if } e_i \textit{ has sublements and/or attributes} \\ A_i \cup t \quad : \textit{if } e_i \textit{ has attributes and/or text content} \\ \textit{empty} \quad : \textit{if } e_i \textit{ has no descendants} \end{cases}$$
- *minOccurrences and maxOccurences specify how many times an attribute or a subelement must appear within an element (the number of occurrences for attributes is 0 or 1, and for elements it varies between 0 and ∞).* ∎

Definition 2. *A reduced XML schema is defined to be an XML schema S = <root, E> where for any element $e \in E$, except the root, the following condition holds: type(e) $\neq$ empty. A reduced XML tree is the XML tree associated to a reduced XML schema.* ∎

Notice that empty elements are mainly used to structure data in a logical way for the user. The reduced XML schema eliminates empty elements as they do not store any data. Thus, *type(e)= integer, string, date, etc.* A reduced XML schema has two major advantages over the source XML schema: (1) it has one node type: the element node, and (2) its hierarchical structure is linearized or its height greatly reduced. Both features allow an optimized comparison of schemas that considers the type and amount of data that can be stored by each leaf node and, ultimately, by the XML data files. The XML reduced tree has only the nodes essential for storing data. Any additional node in a reduced tree means more data is stored in the corresponding XML data file. Two reduced trees are equivalent if their leaf contents are equivalent; and two XML schemas are equivalent if their corresponding reduced tress are equivalent. The equivalence of leaf contents includes element-based similarity (like Cupid [9]) considering name and data type similarity. In addition, our leaf content similarity considers constraints (number of occurrences, AND constraint for sequence, OR constraint for *choice*) for each structure the leaves are part of and constraints (number of occurrences, primary or foreign key) for each leaf.

XML schema has two types of basic structures that can store data: text contents and attributes. There is no developed standard for XML that specifies when an attribute or a text content should be used. Thus, there are many variants of the same schema from this perspective alone. XML elements are classified [4] in two large categories: simple elements and complex elements. Simple elements contain only text data and no attributes or subelements. The term "only text data" refers to any of the simple data types accepted by XML Schema: integer, string, boolean, date, *etc.* Type definitions for elements can use occurrence indicators to change the number of times it is required to appear (default

[4] Mapping *descendent* refers to direct descendents.

is once). Attributes are considered to be of simple type similar to simple elements. The same simple data types apply to both attributes and elements in XML Schemas. Complex elements are classified in four categories [4]: complex text-only (CTO), complex empty (CE), complex subelement-only (CSO), and complex mixed (CM).

4 XML SCHEMA REDUCED TREE

4.1 Nested DTD Notation

XML trees are represented using regular DTD expression operators (?, +, *). "R" denotes *required*[5] structures or nodes where its presence is necessary for clarity. Nested, *sequence* grouped or *choice* alternative elements are represented using round parenthesis ( ). Elements in a tree are ordered if they are part of a *sequence* or unordered when part of an *all* structure. Conversely, attributes are not ordered. To incorporate both aspects in our approach we consider trees to be unordered. This simplification is appropriate because it is important to find equivalent data units, though not necessarily defined in the same order. Thus, *sequence* and *all* structures are considered to be equivalent so they are encoded similarly using commas between elements. Considering all these notations, inspired from DTD, our schema representation is called *a nested DTD*. The nested DTD notation incorporates the advantages of both XML Schema and DTD: (1) it uses compact expressions to define a structure (like DTD), (2) the user can easily identify the tree root so the hierarchy is evident (like XML Schema), and (3) it includes a variety of data types (string, integer, boolean, *etc.*), primary keys, and foreign keys (like XML Schema).

In the XML tree an element with text content is represented by an element and a text node; and an attribute by an attribute node. The text node borrows its label from its element and is nested inside the parent element node. For ease of understanding we append T as a subscript for text nodes and E for element nodes. Using the example (a) from Table 1, $Title$ is an element with $PCDATA$ content that is represented by an element node $Title_E$ and a text node $Title_T$. Conversely, an attribute is represented by its label with A appended to it. Example (b) from Table 1, shows $Title$ as an attribute encoded as $Title_A$. The element $Project$ in this example has no text content so it is represented by an element node $Project_E$ alone.

4.2 A Bottom-Up Approach for XML Schema Reduced Tree Construction

Comparison of XML schema trees has two main disadvantages: (1) three node types: element, attribute, and text and (2) different hierarchical organization. The scope of RA is to simplify the comparison of XML trees by dealing with

[5] The term *required* written in italics refers to mandatory attributes and elements using the XML schema syntax [4].

Table 1. Element, text, and attribute nodes notations

	Data unit	DTD and nested DTD representations
(a)	Element, text	DTD: <!ELEMENT Title (#PCDATA)> Nested DTD: $Title_E(Title_T\#PCDATA)$
(b)	Element Attribute	DTD: <!ELEMENT Project EMPTY> <!ATTLIST Project Title CDATA #IMPLIED> Nested DTD: $Project_E(Title_A\#CDATA)$

a single node type and low height trees. RA eliminates the first disadvantage by transforming all nodes into a single node type. An attribute or a text node requires a parent element so an XML tree that has attributes or text nodes also has elements. The only node type whose existence does not depend on other nodes is the element. Attribute and text nodes are transformed into element nodes preserving the height of the XML tree. The first three rules of RA are strongly connected and must be considered to conserve a single node type.

Reduction Rules Our bottom-up algorithm reduces any XML tree to the minimum number of nodes required to store the same data in the source XML file. RA is based on seven rules. The first three rules called *conversion rules* convert the attribute and text node types of the source structure into an element node, and the last four rules called *elimination rules* eliminate intermediate tree levels.
Rule 1 A text node from a complex text-only element is transformed into an attribute node by borrowing the label of the parent element node[6]. ■

Table 2. Transformations using Rules 1, 2, 3

Rule	Type	Transformations
Rule 1	(a) CTO	$Project_E(Project_T, Title_A) \overset{R1}{\Rightarrow}$ $Project_E(Project_A, Title_A)$
Rule 2	(b) CE	$Project_E(Title_A) \overset{R2}{\Rightarrow}$ $Project_E(Title_E(Title_T))$
	(c) CM	$Project_E(Title_A?, Detail_E(Detail_T)) \overset{R2}{\Rightarrow}$ $Project_E(Title_E?(Title_T), Detail_E(Detail_T))$
Rule 3	(d) Simple	$Title_E(Title_T) \overset{R3}{\Rightarrow} Title_E$

Text nodes share the data type definitions with attributes except they do not have labels. Since we are interested in the equivalence of data that is stored, a text node is represented by an attribute. Thus, a text node is transformed into a

[6] The same final result is obtained if it is transformed into an element (element node and text node).

required attribute node. Table 2 exemplifies Rule 1 applied to the complex text-only element (CTO). The text node $Project_T$ is transformed into the attribute node $Project_A$. Note that for clarity, data types are not specified. By following Rule 1, complex text-only elements become complex-empty elements.

Rule 2 An attribute node is transformed into an element node and a text node. ■

Rule 2 applies to complex empty and mixed element types. A *required* attribute node is transformed into a *required* element formed by an element and a text node. An optional attribute node becomes an optional element. In example (b) from Table 2 the attribute node $Title_A$ becomes the structure formed by an element and a text node $Title_E(Title_T)$. In example (c) the optional attribute $Title_A?$ transfers its operator to the node element $Title_E?$.

By following Rules 1 and 2, the complex text-only, empty, or mixed elements become subelement-only elements. Rule 2 may not appear to serve our purpose of reducing the height of the XML tree. However, this facilitates comparison between two schema structures as there are only two node types: element and text nodes. The attribute node is no longer a node type in the reduced tree.

Rule 3 All text nodes are eliminated. ■

Before Rule 3 is applied text nodes are attached to leaf (simple element) nodes. A leaf node is the only element with a text node. The elimination of text nodes at this point does not create any confusion between an empty (no text content) or non-empty (with text content) element. The concept of leaf element nodes is extended to include the data type of the text node. After Rules 1 and 2 are applied, the XML tree has only two types of nodes: element and text. After Rule 3 is applied there are no more text nodes in the structure and only one node type is used in the tree: the element node (see Table 2 (d)).

Table 3 details how Rules 1, 2, and 3 are applied for each type of element. When a rule does not apply for an element type it is represented using a dash "-". The table shows the reduced tree is formed by simple and subelement-only element nodes. The names of the nodes are X, Y, and Z and the appended letter (A, E, or T) specifies if they are attribute, element, or text nodes.

Table 3. Rules 1, 2, and 3 applied to different element types

Type	Initial representation	Rule 1	Rule 2	Rule 3
Simple	$X_E(X_T)$	-	-	X_E
CTO	$X_E(X_T, Y_A)$	$X_E(X_A, Y_A)$	$X_E(X_E(X_T), Y_E(Y_T))$	$X_E(X_E, Y_E)$
CE	$X_E(Y_A, Z_A)$	-	$X_E(Y_E(Y_T), Z_E(Z_T))$	$X_E(Y_E, Z_E)$
CSO	$X_E(Y_E(Y_T), Z_E(Z_T))$	-	-	$X_E(Y_E, Z_E)$
CM	$X_E(Y_A, Z_E(Z_T))$	-	$X_E(Y_E(Y_T), Z_E(Z_T))$	$X_E(Y_E, Z_E)$

Rule 4 The reduction[7] of a non-repeated element node with non-repeated subnodes requires its optional operator (if any) be transferred (**Rule 4.1**) individually

[7] Node reduction means node elimination.

to child nodes if all subnodes are optional, or (**Rule 4.2**) to the structure that gathers all its child nodes if at least one subnode is *required.* ■

Table 4. Rule 4: Reduction of a non-repeated element with non-repeated subelements

Parent node	Child nodes	Reduction
R	R/?	$X_E(Y_E(Z_E?, V_E)) \overset{R4.1}{\Rightarrow} X_E(Z_E?, V_E)$
?	R,?	$X_E(Y_E?(Z_E?, V_E)) \overset{R4.2}{\Rightarrow} X_E(Z_E?, V_E)?$
?	?	$X_E(Y_E?(Z_E?, V_E?)) \overset{R4.1}{\Rightarrow} X_E(Z_E?, V_E?)$

This rule eliminates non-essential nodes that cannot store data in the XML file. It transfers the parent node's constraint to its children by maintaining the connections between subnodes such that the reduced structure has no information loss (see Table 4). We next analyze two situations when the parent's constraint is allowed to pass and combine directly with its children. Consider Figure 3 that presents two different initial structures that are reduced to the same structure. In the first situation (see Figure 3(a)) both child nodes Z_E and V_E are optional. Eliminating their parent Y_E and transferring its optional constraint to them results in structure $X_E(Z_E?, V_E?)$ with no loss of information. Conversely, in the second situation (see Figure 3(b)), after reduction loses the restriction that V_E can never be present without Z_E. Rule 4 ensures that no such loss of information is allowed.

$$\boxed{\begin{array}{l} \text{(a) } X_E(Y_E?(Z_E?, V_E?)) \Rightarrow X_E(Z_E?, V_E?) \\ \text{(b) } X_E(Y_E?(Z_E, V_E?)) \Rightarrow X_E(Z_E?, V_E?) \end{array}}$$

Fig. 3. Loss of information not allowed by Rule 4

Rule 5 The reduction of a non-repeated element with repeated subelements requires its optional operator (if any) be transferred (**Rule 5.1**) individually to child nodes if all subnodes are optional, or (**Rule 5.2**) to the structure that gathers all its child nodes if at least one subnode is *required.* ■

Table 5. Rule 5: Reduction of a non-repeated element with repeated subelements

Parent node	Child nodes	Reduction
R	+/*	$X_E(Y_E(Z_E+, V_E*)) \overset{R5.1}{\Rightarrow} X_E(Z_E+, V_E*)$
?	+,*	$X_E(Y_E?(Z_E+, V_E*)) \overset{R5.2}{\Rightarrow} X_E(Z_E+, V_E*)?$
?	*	$X_E(Y_E?(Z_E*, V_E*)) \overset{R5.1}{\Rightarrow} X_E(Z_E*, V_E*)$

Rule 5 works similarly to Rule 4 and has no information loss (see Table 5).

Table 6. Rule 6: Reduction of a repeated element with non-repeated subelements

Parent node	Child nodes	Reduction
+	R,?	$X_E(Y_E + (Z_E, V_E?)) \overset{R6.2}{\Rightarrow} X_E(Z_E, V_E?)+$
+	?	$X_E(Y_E + (Z_E?, V_E?)) \overset{R6.1}{\Rightarrow} X_E(Z_E*, V_E*)$
*	R,?	$X_E(Y_E * (Z_E, V_E?)) \overset{R6.2}{\Rightarrow} X_E(Z_E, V_E?)*$
*	?	$X_E(Y_E * (Z_E?, V_E?)) \overset{R6.1}{\Rightarrow} X_E(Z_E*, V_E*)$

Rule 6 The reduction of a repeated element with non-repeated subelements requires its cardinality be transferred (**Rule 6.1**) individually to child nodes if all subnodes are optional, or (**Rule 6.2**) to the structure that gathers all its child nodes if at least one subnode is *required* (see Table 6). ■

A reduction in the case of a repeated parent element with non-repeated subelements must be done more carefully so there is no loss of information (see Table 6). For example, consider the nested structure described by $Company(Em-ployee + (Name, Address))$. Reducing this structure to $Company(Name+, Address+)$ is not the best solution as the constraint of exactly one address for each name is lost; it also accepts interpretations such as "many employees live at the same address" or "many addresses for one employee". Thus, a suitable reduction is to use a repeated sequence such as $Company(Name, Address)+$ that preserves this restriction. Conversely, if all subnodes are optional then the operator * or + is transferred as * to each subelement. For example, the structure $Company(Employee+(Name?, Address?))$ allows independent values for names and addresses without requiring a strong connection between them and is reduced to $Company(Name*, Address*)$. However, if at least one subelement is *required* and the rest are optional, e.g. $Company(Employee+(Name, Address?))$, then for each address there must exist a name. The reduction in this case transfers the operators + or * to the entire sequence as detailed in Table 6.

Rule 7 The reduction of a repeated element with repeated subelements requires its cardinality be transferred (**Rule 7.1**) individually to child nodes if all of them are optional, or (**Rule 7.2**) to the sequence that gathers all its subnodes if at least one subnode is *required* (see Table 7). ■

Table 7. Rule 7: Reduction of a repeated element with repeated subelements

Parent node	Child nodes	Reduction
+	+,*	$X_E(Y_E + (Z_E+, V_E*)) \overset{R7.2}{\Rightarrow} X_E(Z_E+, V_E*)+$
*	+,*	$X_E(Y_E * (Z_E+, V_E*)) \overset{R7.2}{\Rightarrow} X_E(Z_E+, V_E*)*$
+	*	$X_E(Y_E + (Z_E*, V_E*)) \overset{R7.1}{\Rightarrow} X_E(Z_E*, V_E*)$
*	*	$X_E(Y_E * (Z_E*, V_E*)) \overset{R7.1}{\Rightarrow} X_E(Z_E*, V_E*)$

Rule 7 works similarly to Rule 6. To preserve the correlation between *required* subelements, they are kept inside repeated sequences (see Table 7).

Rules 4-7 apply to situations when either all subelements or none are repeated. If a combination of repeated and non-repeated subelements are nested inside a parent element then we must ensure that a lossless reduction is applied. There are situations when two rules that contradict each other must be applied to reduce the outer element. They contradict because a rule allows expression operators to be transferred to subnodes while the other requires operators to be transferred to the sequence. The scope of the Reduction Algorithm is to reduce levels in the XML tree without losing any data node. In these situations if we allow the operators to be transferred to some subnodes, then some connections between them are lost. Thus, the expression operator of the outer element must be transferred to the entire sequence. Figure 4 details two situations. In (a) Rule 6.1 is applied to reduce Y_E for subnode Z_E and suggests the operator $+$ be passed to Z_E and combine with its operator ?. However, Rule 7.2 applied to Y_E and V_E suggests the operator $+$ be attached to the sequence grouping Z_E and V_E. The lossless reduction of this structure reduces Y_E by transferring its cardinality to the sequence. A similar situation is presented in Figure 4(b).

$$\boxed{\begin{array}{l} \text{(a) } X_E(Y_E+(Z_E?,V_E+)) \overset{R6.1,R7.2}{\Rightarrow} X_E(Z_E?,V_E+)+ \\ \text{(b) } X_E(Y_E+(Z_E,V_E*)) \overset{R6.2,R7.1}{\Rightarrow} X_E(Z_E,V_E*)+ \end{array}}$$

Fig. 4. Combining two rules

All rules detailed in this section were exemplified using sequences. They are applied similarly to the *choice* structures by considering that each alternative within this structure has the operator *required* if no operator is specified.

The Reduction Algorithm (RA) The bottom-up Reduction Algorithm contains two parts (see Figure 5). The first part prepares the XML schema for reduction by normalizing and modifying it to use only one node type: elements. The normalization process is based on the algorithm developed by Arenas and Libkin [1] that obtains an XNF normal form for XML Schema following the "standard treatment" of BCNF. The concept of normal form transferred to XML context XNF refers to a well-designed schema with non-redundant information that prevents update anomalies in the XML data file. Further, the XML Schema in XNF is a minimal form as redundant element and attribute definitions are removed. Finally, the Reduction Algorithm eliminates the text and attribute nodes from the structure by transforming them into element nodes through Rules 1, 2, and 3.

The second part of our algorithm reduces nodes by using Rules 4, 5, 6, and 7 depending on the type of parent and child nodes (repeated, optional, *etc.*). These steps generate a reduced structure that preserves the initial constraints of the XML schema. Transformation rules inspired from Lee *et al.* [8] and detailed in Table 8 are used to combine expression operators in Rules 4, 5, 6, and 7.

The result of RA is a reduced XML tree formed by leaf element nodes (element with data type). Rule 3 has a deeper meaning after Rules 4-7 are applied and intermediate element nodes are eliminated. Rule 3 creates the key nodes that will be compared to determine equivalent schemas.

```
1. Part 1: Prepare the XML schema for reduction
   (a) Convert the XML schema into an XNF using Arenas and Libkin's
       algorithm [1] (normalization).
   (b) Represent the XML schema using the DTD nested notation.
   (c) Apply Rule 1 to transform text nodes from text-only elements into
       attribute nodes.
   (d) Apply Rule 2 to transform attribute nodes into element and text
       nodes. (It eliminates the attribute nodes.)
   (e) Apply Rule 3 to eliminate text nodes.
2. Part 2: Reduce the XML tree starting from the leaf nodes going up
   to the tree root by applying Rules 4, 5, 6, and 7 according to each
   situation.
```

Fig. 5. The Reduction Algorithm

Table 8. Combination of Operators

Operator 1	Operator 2	Result
R	R	R
?	R/?	?
?	+/*	*
+	R/+	+
+	*	*
*	R/ *	*

4.3 Example Using RA

This section presents an example that illustrates our approach. Consider the example from Figure 2. The first part of the Reduction Algorithm prepares the source XML schema by checking that it is in XNF normal form. The structure must then be transformed from a DTD into our notation. Recall that we append T as subscript for text nodes, A for attribute nodes, and E for element nodes.

$$company2_E(employee_E + (eid_A\#integer, sin_E(sin_T\#integer), name_E (name_T\#string), address_E * (address_T\#string), dateOfBirth_E? (dateOfBirth_T\#date), projects_E?(project_E + (pid_A?\#integer, description_E?(description_T\#string), manager_E(manager_T\#string) |location_E(location_T\#string), task_E + (task_T\#string, date_A\#date)))))$$

Rule 1 is applied to transform the text node $task_T$ from the text-only element $task_E$ into an attribute node. Data types are not included in the following explanations for clarity reasons.

$$\overset{R1}{\Rightarrow} company2_E(employee_E{+}(eid_A, sin_E(sin_T), name_E(name_T), address_E{*}(address_T), dateOfBirth_E?(dateOfBirth_T), projects_E?(project_E{+}(pid_A?, description_E?(description_T), manager_E(manager_T)|location_E(location_T), task_E + (task_A, date_A)))))$$

Rule 2 transforms attribute nodes into element and text nodes. Thus, the structure does not contain any attribute nodes anymore.

$$\overset{R2}{\Rightarrow} company2_E(employee_E + (eid_E(eid_T), sin_E(sin_T), name_E name_T), address_E * (address_T), dateOfBirth_E?(dateOfBirth_T), projects_E?(project_E{+}(pid_E?(pid_T), description_E?(description_T), manager_E(manager_T)|location_E(location_T), task_E + (task_E(task_T), date_E(date_T))))))$$

Rule 3 next eliminates all text nodes.

$$\overset{R3}{\Rightarrow} company2_E(employee_E{+}(eid_E, sin_E, name_E, address_E{*}, dateOfBirth_E?, projects_E?(project_E + (pid_E, description_E?, manager_E|location_E, task_E + (task_E, date_E)))))$$

Since they are all element nodes at this point, we drop the suffix E from the element nodes and we apply Rule 6.2 twice from bottom-up to reduce the nodes *task*+ and *project*+, respectively.

$$\overset{R6.2}{\Rightarrow} company2(employee + (eid, sin, name, address{*}, dateOfBirth?, projects?(project{+}(pid, description?, manager|location, (task, date){+}))))$$

$$\overset{R6.2}{\Rightarrow} company2(employee + (eid, sin, name, address{*}, dateOfBirth?, projects?(pid, description?, manager|location, (task, date){+}){+}))$$

Rules 4.2 and 5.2 reduce the node *projects*?. The ? operator is transferred to the sequence of the projects' subnodes. The ? operator next combines with + and it results $*$.

$$\overset{R4.2,R5.2}{\Rightarrow} company2(employee + (eid, sin, name, address{*}, dateOfBirth?, (pid, description?, manger|location, (task, date){+}){*}))$$

Rules 6.2 and 7.2 are applied to reduce the node *employee*+.

$$\overset{R6,R7}{\Rightarrow} company2(eid, sin, name, address{*}, dateOfBirth?, (pid, description?, manger|location, (task, date){+}){*}){+}$$

No further reduction is done within Part 2 of the Reduction Algorithm. The resulted structure is reduced in Part 3 by moving the expression operators inside the sequences.

$$\Rightarrow company2(eid, sin, name, address{*}, dateOfBirth?, (pid, description?, manger|location, task{+}, date{+}){*}){+}$$

$$\Rightarrow company2(eid, sin, name, address{*}, dateOfBirth?, pid{*}, description{*}, manger * |location{*}, task{*}, date{*}){+}$$

$$\Rightarrow company2(eid{+}, sin{+}, name{+}, address{*}, dateOfBirth{*}, pid{*}, description{*}, manger * |location{*}, task{*}, date{*})$$

5 ALGORITHM EVALUATION

5.1 Real World Example

We searched for XML Schemas used in other approaches dealing with XML schema equivalence and found only DTDs. Thus, we decided to select a DTD and transform it into XML Schema. To exemplify RA on a real world example we use the schema found in pise.dtd[8] from the molecular biology area. We transform this DTD schema into an XML Schema[9] by creating a tree structure and including additional data types (i.e. boolean) as suggested by its authors. The pise structure has a total of 85 nodes containing 63 leaves, 12 levels from which 7 are determined by arbitrary organizational nodes and 5 by repeatable sequences and choice structures. The schema at the end of Part 3 of RA contains one level with 63 nodes; at the end of Part 2 it contains 6 levels (the leaves' level and 5 additional levels to preserve the structural organization created by repeatable sequences and *choices*). The total number of nodes in Part 2 is 63. This example demonstrates that RA decreases significantly the height of XML trees.

Part 3 of RA simplifies the comparison of two schemas even more. To identify if another schema called test.xsd is from the same domain as pise.xsd 63 leaves from the latter are compared with the former's leaves. This is a linear comparison that does not consider the number of levels in the hierarchies. Suppose the initial non-reduced schema of test.xsd has 10 levels, 7 determined by arbitrary organizational nodes and 3 by repeatable structures, with 70 nodes containing 50 leaves. If the source schemas were compared using any algorithm from Related work then 85 nodes from pise.xsd split into 12 levels must be compared with 70 nodes from test.xsd split into 10 levels. This implies that 85 paths from one pise.xsd some as long as 12 levels must be compared with 70 paths 10 levels long from the test.xsd schema. Using RA we compare 63 nodes that are part of at most 5 nested structures with cardinalities with 50 nodes that are part of at most 3 nested repeatable or optional structures. Through this perspective we consider that RA simplifies significantly the complexity of structural equivalence algorithms.

5.2 Algorithm Complexity

The transformation of the source XML Schema into a nested DTD schema is performed in $Max(O(m), O(pn))$ time and includes three parts: (1) reading the total number m of nodes in $O(m)$ time, (2) associating primary keys p with corresponding leaves n done in $O(pn)$, and (3) associating foreign keys f with corresponding nodes and primary keys done in $O(fn)$.

The nested DTD schema is then transformed using Conversion Rules in $O(n)$ time. Reductions according to Elimination Rules are performed in $O(kn)$,

[8] Created by Catherine Letondal from Institut Pasteur and available at www.visualgenomics.ca/gordonp/xsd.

[9] Schema pise.xsd is available at www.cpsc.ucalgary.ca/∼duta/SchemaEquiv/pise.xsd.

where k is the number of levels determined by repeatable structures (*choice* and sequence). Reduction of schemas creates the framework for a new structural schema equivalence algorithm which we anticipate is executed in less than quadratic time. The new schema equivalence algorithm applies to schema trees of similar or different structural organization as those exemplified by Figures 1 and 2.

5.3 Discussion

RA is based on the flexibility of XML Schema to express cardinalities (+, * or ?) in different ways. Consider the definition of *employee* from Figure 1. The structure *employee*+ includes the cardinality in the element definition written in XML Schema $< element\ name = "employee"\ maxOccurs = "unbounded" >$. The same structure can be expressed attaching the cardinality to the sequence that defines the element *employee*: $< sequence\ maxOccurs = "unbounded" >$. The latter expression is translated in nested DTD as
$employee(eid \mid sin,\ name,\ pid*)+$.

Note that XML schema allows nested sequences and/or *choice* structures to be defined without requiring a parent element for each sequence or *choice*. The transfer of parent element cardinality to its inner sequence (or *choice*) together with parent elimination simplifies the initial schema without changing its hierarchy. [10] We are currently working on a formal proof of our algorithm correctness to demonstrate (1) the final reduced tree is the minimum hight tree that preserves cardinality constraints and (2) our set of seven reduction rules is minimum and the omission of any rule determines a non-optimum reduced tree.

6 CONCLUSION AND FUTURE WORK

Our approach transforms XML schemas into minimum structures capable of storing the same information. The Reduction Algorithm is an important step in preparing schemas for an optimum comparison with no loss of information. RA preserves the cardinality of different parts of schema's initial hierarchy; however, it eliminates intermediate nodes whose only scope is to make the structure more readable. RA defines the framework for a flexible comparison of XML Schemas that do not share the same hierarchical organization. We are conducting further research to address structural equivalence for XML schemas using reduced trees.

References

1. M. Arenas and L. Libkin. A normal form for xml documents. *ACM Transactions on Database Systems (TODS)*, 29(1):195 – 232, March 2004.
2. E. Bertino, G. Guerrini, and M. Mesiti. A matching algorithm for measuring the structural similarity between an xml document and a dtd and its applications. *Journal of Information Systems*, 29(1):23–46, 2004.

[10] For more details on this aspect please refer to [4].

3. J. Boyer. Canonical xml version 1.0, w3c recommendation. March 2001.
4. W. W. W. Consortium. Xml schema part 0, 1, and 2. October 2004.
5. H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the 2nd Int. Workshop on Web Databases*, 2002.
6. H. H. Do and E. Rahm. Coma - a system for flexible combination of schema matching approaches. In *Proceedings of the 28th VLDB Conference*, pages 610–621, August 2002.
7. P. U. C. S. Laboratory. *WordNet. An Electronic Lexical Database*. MIT Press, Reading, 1998.
8. M. L. Lee, L. H. Yang, W. Hsu, and X. Yang. Xclust: Clustering xml schemas for effective integration. In *Proceedings of the 11th ACM International Conference on Information and Knowledge Management*, pages 292–299, November 2002.
9. J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the 27th VLDB Conference*, pages 49–58, 2001.
10. A. Nierman and H. Jagadish. Evaluating structural similarity in xml documents. In *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)*, June 2002.
11. A. Salminen and F. W. Tompa. Requirements for xml document database systems. In *Proceedings of the 2001 ACM Symposium on Document Engineering*, pages 85–94, 2001.