XML Schema Reduction Algorithm

Angela C. Duta, Ken Barker, Reda Alhajj
University of Calgary, Department of Computer Science
2500 University Drive N.W. Calgary, Alberta Canada, T2N 1N4
{duta, barker, alhajj}@cpsc.ucalgary.ca

June 2004

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 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 ([4], [5], [6]) done in this area but we believe there is still additional work to be done aimed at 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 features of the XML structure. 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 schema for structural matching based on the equivalent leaves content rather than hierarchical context and vicinities. A *leaf content*[1] 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 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]Note that a *leaf content* is different from the *element content* formed by text and attributes used by W3C [8].

1.3 Paper Organization

The balance of this paper is organized as follows. Section 2 presents two DTD examples that could be considered equivalent. Section 3 briefly discusses several developed methods to evaluate XML schema equivalence. Section 4 defines several concepts used in this paper while Section 5 introduces our approach. This paper draws some conclusions in Section 6.

2 Motivating Examples

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 XML schemas and decide if they are equivalent. This paper presents a reduction method for XML schemas to be used before the equivalence algorithm is executed.

```
<!ELEMENT company1 (employee+, project*, task+)>
<!ELEMENT employee (eid | sin, name, pid* | task_name)>
<!ATTLIST employee address CDATA #REQUIRED>
<!ELEMENT project (pid, description, budget | manager | location )>
<!ELEMENT task(task_name, date)>
---
keys: project.pid, task.task_name
keyref: employee.pid to project.pid, employee.task_name to task.task_name
```

Figure 1: Repeated employee, project, and task elements

3 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

```
<!ELEMENT company2 (employee+)>
<!ELEMENT employee (sin, name, address*, dateOfBirth?, projects?)>
<!ATTLIST employee eid CDATA #REQUIRED>
<!ELEMENT projects (project+)>
<!ELEMENT project (description?, manager | location, task+)>
<!ATTLIST project pid CDATA #REQUIRED>
<!ELEMENT task+ (#PCDATA)>
<!ATTLIST task date CDATA #REQUIRED>
----
keys: employee.eid, project.pid
keyref: project.manager to employee.eid
```

Figure 2: Nested structure of employee, project and task elements

provided a starting point to address this problem by defining canonical forms for XML [3]. Salminen and Tompa [7] 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 [6] 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.* [5] propose a method for scalable integration of DTDs. The method is based on two steps (1) finding similar DTDs, and (2) generating DTD clusters. A key aspect to this work is finding similar DTDs. 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. Ontology similarity [5] (a component of semantic similarity) is based on finding similar labels or synonyms (using WordNet Java API) 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, based on them, 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.

An evaluation of the most recent approaches in schema matching for relational or XML models is available [4]. Do *et al.* [4] present an objective comparison of different approaches that use different effectiveness evaluation measures. The conclusion of this work is that it is still unclear how effective are each of the studied matching tools. This is because there has been no uniform system used to evaluate the effectiveness of each tool. Do *et al.* [4] suggest creation of a schema matching benchmark containing real-length XML schemas to be used to test, evaluate, and compare future approaches. This proposed framework does not apply to our approach which prepares schemas for matching.

4 Algorithm Framework

4.1 Definitions

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¿* and four mappings *type, descendent, minOccurences,* and *maxOccurences*, as follows:

- E is a finite set of elements.
- A is a finite set of attributes
- *root* is the only element for which all other elements from E are direct or indirect descendants.
- *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 ∞).
- *type* is a mapping from E or A to the XML data type of each element and attribute. Consider e an element in E and a an attribute in A, then the mappings are defined as follows:
  - *type(e)= empty, integer, string, date, etc.*
  - *type(a)= integer, string, date, etc.* ■

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

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. 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 transformed into a linear one. 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.

4.2 XML Schema Definition, Elements, and Attributes

DTDs (Document Type Definition) seem to be the most popular XML schema representation with the more recent representation XML Schema Definition (or XML Schema) following closely. XML Schema is a more verbose definition language to DTD as it uses numerical and character data types for elements and attributes [8], which is important information in determining if two schemas are equivalent. In this paper we use DTD to detail examples of XML schemas as it is more compact but our algorithm is based on XML Schema because it includes a larger variety of data types and features (e.g. primary and foreign keys constraints for attributes and elements)[2].

XML schema is also a richer database schema as it has two types of basic structures that can store data: elements and attributes. There is no developed standard for XML that specifies when an attribute or an element should be used. Thus, there are many variants of the same schema from this perspective alone. XML elements are classified [8] 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.* Complex elements are elements of complex type and are classified in four categories [8]:

- complex text-only (CTO) elements with several attributes and restrictions or

[2] As a result we take a few liberties with the DTD nomenclature in our example but will indicate these where they occur.

extensions to the base data types;

- complex empty (CE) elements with no content but with one or more attributes;
- complex (sub)element-only (CSO) elements[3] with no content but with one or more elements;
- complex mixed (CM) elements whose complex type definition allows text and attributes, or attributes and elements.

Complex type definitions for elements with subelements can use occurrence indicators to change the number of times a subelement is required to appear (default is once). Attributes are considered to be of simple type similar to simple elements. The same set of simple data types applies to both attributes and elements in XML Schemas.

5 XML Schema Reduced Tree

5.1 Nested DTD Notation

XML trees are represented using regular DTD expression operators (?, +, *). "R" denotes *required*[4] 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 within the scope of our approach as it is important to find equivalent data units,

[3]W3C (see [8]) uses the term (sub)elements-only element to represent elements with no content or attributes. In the interest of readability the term (sub)element-only element is used in this paper.

[4]The term *required* written in italics refers to mandatory attributes and elements using the XML Schema syntax [8].

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*.

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 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. The 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.

A complex text-only element with a text node and an attribute is different than a complex empty element with one attribute as it can store additional data in the text node (see Table 2). Similarly, a complex subelement-only or a mixed element has each subelement represented as the union of an element and a text node. Each subelement is a text-only, empty, subelement-only or mixed element, repeated or not. A mixed type element is not allowed to have a text content but it can have an unlimited number of elements and/or attributes. Table 2 details the schema encoding and the nested DTD representation for different complex types.

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

Table 1: Element, text, and attribute nodes notations

Type	Representations		Data units
(a) CTO	DTD	<!ELEMENT Project (#PCDATA)>	Element, Text
		<!ATTLIST Project Title CDATA #REQUIRED>	Attribute
	Nested DTD	$Project_E(Project_T, Title_A)$	
(b) CE	DTD	<!ELEMENT Project EMPTY>	Element
		<!ATTLIST Project Title CDATA #REQUIRED>	Attribute
	Nested DTD	$Project_E(Title_A)$	
(c) CSO	DTD	<!ELEMENT Project (Title, Description)>	Elements
		<!ELEMENT Title (#PCDATA)>	Element, Text
		<!ELEMENT Detail (#PCDATA)>	Element, Text
	Nested DTD	$Project_E(Title_E(Title_T), Detail_E(Detail_T))$	
(d) CM	DTD	<!ELEMENT Project (Description)>	Elements
		<!ATTLIST Project Title CDATA # IMPLIED>	Attribute
		<!ELEMENT Detail (#PCDATA)>	Element, Text
	Nested DTD	$Project_E(Title_A?, Detail_E(Detail_T))$	

Table 2: DTD and nested DTD representation for complex type elements

5.2 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 a single node type and low hight 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; thus, 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. However, text nodes are first changed into attribute nodes and then attributes are transformed into elements. If a text node were transformed directly into an element (which is represented by an element node and a text node), it would increase the XML tree hight with no evident benefit. Conversely, transforming a text node into an attribute node preserves the hight of the XML tree. The first three rules of RA are strongly connected and must be considered together serving the purpose of using a single node type. RA deals with subjective hierarchical organizations by eliminating higher level nodes and keeping only lower level ones (see Figures 1 and 2 for two examples of different hierarchical organizations).

5.2.1 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 six rules. The first three rules convert the attribute and text node types of the source structure into an element node, and the last three 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. ■

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	$Project_E(Title_A) \overset{R2}{\Rightarrow} Project(Title_E(Title_T))$
	(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	$Title_E(Title_T) \overset{R3}{\Rightarrow} Title_E$

Table 3: Transformations using Rules 1, 2, 3

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 *required* attribute node. Table 3 exemplifies Rule 1 applied to the complex text-only element from Table 2 (a). The text node $Project_T$ is transformed into the attribute node $Project_A$. 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 3 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 node type in the reduced tree.

Rule 3 All text nodes are eliminated. ■

Before Rule 3 is applied the 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 3 (d)).

Table 4 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.

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	$X_E(Y_A, Z_E(Z_T))$	-	$X_E(Y_E(Y_T), Z_E(Z_T))$	$X_E(Y_E, Z_E)$

Table 4: Rules 1, 2, and 3 applied to different element types

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

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,?	$X_E(Y_E?(Z_E?, V_E)) \overset{R4.2}{\Rightarrow} X_E(Z_E?, V_E)?$
?	?	$X_E(Y_E?(Z_E?, V_E?)) \overset{R4.1}{\Rightarrow} X_E(Z_E?, V_E?)$

Table 5: Rule 4: Reduction of a non-repeated element with non-repeated subelements

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 5). 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 looses the restriction that V_E can never be present without Z_E. Rule 4 ensures that no such loss of information is allowed.

(a) $X_E(Y_E?(Z_E?, V_E?)) \Rightarrow X_E(Z_E?, V_E?)$
(b) $X_E(Y_E?(Z_E, V_E?)) \Rightarrow X_E(Z_E?, V_E?)$

Figure 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*. ■

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

Table 6: Rule 5: Reduction of a non-repeated element with repeated subelements

Rule 5 works similarly to Rule 4 and has no information loss (see Table 6).

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,?	$X_E(Y_E * (Z_E, V_E?)) \overset{R6.2}{\Rightarrow} X_E(Z_E, V_E?)*$
*	?	$X_E(Y_E * (Z_E?, V_E?)) \overset{R6.1}{\Rightarrow} X_E(Z_E*, V_E*)$

Table 7: Rule 6: Reduction of a repeated element with non-repeated subelements

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 7). ■

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 7). For example, consider the nested structure described by $Company(Employee+(Name, Address))$. 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 7.

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 8). ■

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

Table 8: Rule 7: Reduction of a repeated element with repeated subelements

Rule 7 works similar to Rule 6. To preserve the correlation between *required* subelements, they are kept inside repeated sequences (see Table 8).

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 loosing 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).

(a) $X_E(Y_E + (Z_E?, V_E+)) \stackrel{R6.1,R7.2}{\Rightarrow} X_E(Z_E?, V_E+)+$
(b) $X_E(Y_E + (Z_E, V_E*)) \stackrel{R6.2,R7.1}{\Rightarrow} X_E(Z_E, V_E*)+$

Figure 4: Combining two rules

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

5.2.2 The Reduction Algorithm (RA)

The bottom-up Reduction Algorithm contains three parts (see Figure 5). The first part prepares the XML schema for reduction by normalizing it 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 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. Unfortunately, it has one drawback: it may contain repeatable or optional sequences. Two reduced schemas that have similar nodes but in different sequences may not seem equivalent. Part 3 eliminates sequences by transferring expression operators (?, +, *) to inner elements. Thus, a wider category of schemas are qualified as equivalent that have similar nodes and cardinalities but not necessarily nested similarly. Sequences are eliminated and expression operators are transferred to the inner elements in Part 3 using the DTD transformation rules inspired from Lee *et al.* [5] and detailed in Table 9.

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.

3. Part 3: Eliminate sequences by transferring the sequence multiplicator (+, *) or optional indicator ? to each node in the sequence.

Figure 5: The Reduction Algorithm

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

Table 9: Operators combination in Part 3 of the Reduction Algorithm

5.3 Example

This section is dedicated to 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.

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

$date_A)))))$

Rule 1 is applied to transform the text node $task_T$ from the text-only element $task_E$ into an attribute node.

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

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 $*$.

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

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

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

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.

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

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

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

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. An optimum comparison of XML schemas must identify files that refer to the same entities but not necessary with the same hierarchical organization. We are conducting further research to address equivalence for XML schemas using reduced trees.

References

[1] Arenas, M. and Libkin, L. *A normal form for XML documents* in Proceedings of PODS 2002, pages 85-96, 2002.

[2] Bertino, E., Guerrini, G., Mesiti, M. *A matching algorithm for measuring the structural similarity between an XML document and a DTD and its applications* in Information Systems, Vol. 29, Issue 1, pages 23-46, 2004.

[3] Boyer, J. *Canonical XML Version 1.0, W3C Recommendation*, White Paper, 15 March 2001, http://www.w3.org/TR/xml-c14n.

[4] Do H.-H., Melnik, S. and Rahm, E. *Comparison of schema matching evaluations* In Web, Web-Services and Database Systems, Lecture Notes in Computer Science, Vol. 2593, pages 221-237, 2003.

[5] Lee, M.L., Yang, L.H., Hsu, W., Yang, X. *XClust: Clustering XML Schemas for Effective Integration* in Proceedings of the eleventh international conference on Information and knowledge management, Virginia USA, pages 292-299, 2002.

[6] Madhavan, J., Bernstein, P.A., Rahm, E. *Generic Schema Matching with Cupid* in Proceedings of the 27th VLDB Conference, pages 49-58, 2001.

[7] Salminen, A. and Tompa, F.Wm. *Requirement for XML Document Database Systems* in Proceedings of the 2001 ACM Symposium on Document Engineering, pages 85-94, Atlanta, Georgia, USA, 2001.

[8] World Wide Web Consortium, *XML Schema Part 0, 1, and 2*, May 2001.