Know your Limits: Enhanced XML Modeling with Cardinality Constraints

Sebastian Link **Thu Trinh**

Department of Information Systems, Information Science Research Centre
Massey University, Palmerston North, New Zealand
E-mail: [S.Link,T.Trinh]@massey.ac.nz

Abstract

XML Schema supports the specification of occurrence constraints by declaring values for its *min*/*max*Occurs attributes. These constraints are structural in the sense that they restrict the cardinality of children elements independently of the data that is present in their subtrees. Thus, occurrence constraints do not address any data semantics and are therefore only of limited interest to data modelling.

In this tutorial we first survey the benefits of introducing semantic cardinality constraints into the framework of XML modeling languages: enhanced modeling capabilities, increased levels of data consistency and integrity, additional techniques for XML query optimization and query rewriting, prediction of the number of XML query answers, updates and encryptions.

Secondly, we survey different techniques of representing many-to-many relationship types within XML. In particular, we focus on the preservation of key, foreign key and participation constraints.

1 Introduction

The eXtensible Markup Language XML (Bray, Paoli, Sperberg-McQueen, Maler & Yergeau n.d.) has evolved to the de-facto standard for information/data representation and exchange on the Internet and elsewhere. This trend is due to the high syntactic flexibility XML provides. Consequently, an increasing number of users find it very simple to create data in XML or an XML dialect, e.g., the familiar HTML, NITF (in the news industry), WeatherML (for weather information), CellML (in bioinformatics), WML (for mobile phones) and XML Pay (for Internet-based payments). There has been wide consensus that XML documents/data should obtain the same type of management functionalities as conventional data received from relational database management systems. The essential difference between XML data and traditional data (e.g. relational data) is the extra structural, tree-like relationships between the various elements of an XML data source. This complex structure makes it rather difficult for the database community to meet the increasingly important challenge of providing full-fledged tools that can store, manage and process this data in its native format. In particular, XML has not been primarily designed to serve as a data model, and thus does not carry the semantics of its data. XML schema languages and integrity constraints can enhance the semantic capabilities of XML in the sense that they permit only those XML databases that are considered meaningful for the application at hand. For almost all previously studied classes of XML constraints (Arenas & Libkin 2004, Buneman, Davidson, Fan, Hara & Tan 2002, Buneman, Fan, Siméon & Weinstein 2001, Buneman, Fan & Weinstein 2000, Fan & Libkin 2002, Fan & Siméon 2003, Hartmann & Link 2003, Hartmann & Trinh 2006, Hartmann, Link & Trinh 2007, Vincent, Liu & Liu 2004) the intricate structure of XML data results in decision problems that are, in general, intractable. It remains a major challenge to find expressive classes of XML constraints that can be efficiently reasoned about (Fan 2005, Fan & Libkin 2002, Fan & Siméon 2003, Suciu 2001, Vianu 2003).

Cardinality constraints are naturally exhibited by XML data since they represent restrictions that occur in everyday practice. They can express more properties than other XML constraints (Arenas & Libkin 2004, Arenas & Libkin 2005, Buneman et al. 2002, Buneman, Davidson, Fan, Hara & Tan 2003, Fan & Libkin 2002, Fan & Siméon 2003, Vincent et al. 2004). Moreover, cardinality constraints have a direct impact on various applications including schema design, query optimization/rewriting, efficient storing/updating, data exchange/integration, and consistent query answering. Commercial pressure and insufficient time resources force many database researchers to narrow their investigations to notions that have only a limited expressiveness and applicability. Hence, many characteristics of XML data remain unaddressed and most applications underexploited. Cardinality constraints have the potential to make a long-lasting contribution to database technologies. Such a contribution, however, requires tractable reasoning capabilities for expressive constraints.

In this tutorial we will identify cardinality constraints as a highly useful and natural class of XML constraints. In order to make effective use of this class in various XML applications the associated satisfiability and implication problems must be studied. Cardinality constraints are defined independently from any specification such as a DTD (Bray et al. n.d.) or XSD (Thompson, Beech, Maloney & Mendelsohn n.d.), and are based on the representation of XML data as trees. This data model is commonly used by DOM (Apparao et al n.d.), XSL (Kay n.d.), and XML Schema (Thompson et al. n.d.). Figure 1 shows such a representation in which nodes are annotated by their type: E for element, A for attribute, and S for text (PCDATA).

Cardinality constraints are defined in terms of path expressions, and restrict the number of nodes that have the same (complex) values on some selected sub-

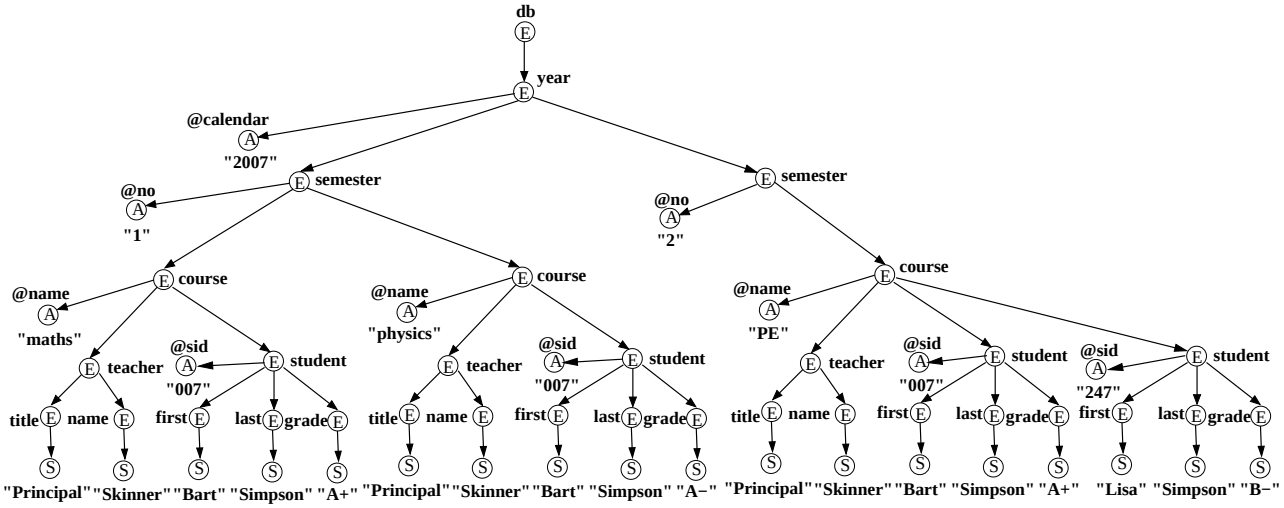

Figure 1: An XML tree

nodes. Such restrictions can be specified either for the entire document or relative to selected context nodes. For the XML tree T in Figure 1 the following cardinality constraint may be specified for the subtrees rooted at *semester*-nodes: each *student/@sid*-value that occurs under some *course*-node actually occurs as a *student/@sid*-value under between two and four *course*-nodes in the same *semester*-subtree. In other words, each student who is studying in a certain semester must enrol in at least two and may enrol in up to four courses. Another cardinality constraint that may be specified for the subtrees rooted at *year*-nodes is the following: if there are *semester*-children then there must be precisely two. While T violates the first constraint it does satisfy the second.

Students may want to query the database for their grades within a particular year using their cell phone, but may only choose this service in case the costs are reasonable. The service provider, on the other hand, wants to invoke its services only if they are paid for. A database management system capable of reasoning about cardinality constraints is able to foresee that the number of answers to the following XQuery (Boag, Chamberlin, Fernández, Florescu, Robie & Siméon n.d.) query is at most eight:

```
for $s in doc("enrol.xml")/
    year[@calendar="2007"]//course/student
where $s/@sid="007"                                  (1)
return ⟨grade⟩{$s/grade}⟨/grade⟩
```

Hence, without querying the database itself the service provider can inform the student of the maximal costs that will be charged for this service, and decide accordingly.
Suppose the database designer has specified the additional cardinality constraint that in each year a teacher can teach up to three different courses. Applying reasoning techniques an XML query optimiser can transform the XQuery query

```
for $c in doc("enrol.xml")/
    year[@calendar="2007"]/semester/course
where $c/student/@sid="247" and
    $c/teacher="Principal Skinner"
return ⟨course⟩{$c/@name}⟨/course⟩
```

into the equivalent query

```
for $c in doc("enrol.xml")/
    year[@calendar="2007"]/semester/course
where $c/teacher="Principal Skinner" and
    $c/student/@sid="247"
return ⟨course⟩{$c/@name}⟨/course⟩
```

which performs better due to the smaller selectivity of course nodes based on teachers rather than students.

The idea of cardinality constraints is not new in database practice: they are commonly known as cardinality or participation constraints in conceptual database design, in particular Entity-Relationship modelling (Hartmann 2001). To the best of our knowledge cardinality constraints have not yet been considered in XML. This is to some degree surprising since these constraints occur naturally in XML data and are very useful for many XML applications as the examples above already indicate. On the other hand the generalisation of constraints to XML is challenging (Fan 2005), in particular the identification of natural classes that can be reasoned about efficiently.

Topics covered. We review cardinality constraints in the context of XML (Hartmann & Link 2007*a*). We believe that these constraints are naturally exhibited by XML data and a thorough study of these constraints can significantly improve the modeling capabilities of XML. Moreover, they generalise both the class of XML keys (Buneman et al. 2002, Hartmann & Link 2007*b*) and several classes of cardinality constraints studied in the context of semantic data models (Hartmann 2001, Lenzerini & Nobili 1990, Liddle, Embley & Woodfield 1993, Thalheim 1992).

In order to take full advantage of these constraints in various XML applications, it is necessary to reason about them. Similarly to many other classes of XML constraints reasoning about cardinality constraints is intractable in general.

We identify *numerical keys* as a large tractable subclass of cardinality constraints. They permit the specification of upper bounds on the number of nodes with value-equal subnodes. In particular, numerical keys subsume XML keys (Buneman et al. 2002, Hartmann & Link 2007*b*) where the upper bound is always fixed to 1.

Unlike many other classes of XML constraints, numerical keys can be reasoned about efficiently. Specifically, any finite set of numerical keys is finitely satisfiable, and the (finite) implication of numerical keys is finitely axiomatisable. Furthermore, numerical key

implication is decidable in time quadratic in the size of the numerical keys given. Therefore, numerical keys are more expressive than XML keys but the time-complexity of their associated implication problems is the same (Hartmann & Link 2007*b*, Hartmann & Link 2007*a*).

We demonstrate the applicability of numerical keys for many XML applications. In the presence of these constraints upper bounds on the number of query answers for XPath (Clark & DeRose n.d.) and XQuery queries can be predicted. In the same spirit one can obtain upper bounds on the number of updates using the XQuery update facility (Chamberlin, Florescu & Robie n.d.) or on the number of en- and decryptions using XML encryption (Imamura, Dillaway & Simon n.d.). We also show how to utilise cardinality constraints in order to define XML views that allow us to process the most common types of updates and queries efficiently.

Finally, we survey different techniques of representing many-to-many relationship types within XML utilising occurrence and cardinality constraints. In particular, we focus on the preservation of key, foreign key and participation constraints.

In summary, we believe that cardinality constraints and numerical keys form natural classes of constraints that can be utilised effectively by XML designers and XML applications.

Related Work. Constraints have been extensively studied in the context of the relational data model (Thalheim 1991). Dependencies have also been investigated in nested data models, see for instance (Hara & Davidson 1999, Hartmann, Link & Schewe 2006). For a brief survey on recent work on XML constraints see (Fan 2005) which also includes many more references. Cardinality constraints were only recently introduced into the context of XML (Hartmann & Link 2007*a*). They extend the class of XML keys (Buneman et al. 2002, Hartmann & Link 2007*b*) as well as cardinality constraints studied in the context of semantic data models (Hartmann 2001, Lenzerini & Nobili 1990, Liddle et al. 1993, Thalheim 1992).

The *minOccurs* and *maxOccurs*-attributes of XML Schema (Thompson et al. n.d.) restrict the number of a node's children but do this independently from the data that is actually occurring. In sharp contrast numerical constraints restrict the number of nodes in XML subtrees based on the data that can be found on selected subnodes.

Organisation. We define cardinality constraints for XML in Section 2 based on the common XML tree model. It is shown that reasoning about numerical constraints is computationally intractable in general. Subsequently, we identify numerical keys as a tractable subclass. In Section 3 we provide (i) a finite axiomatisation , and (ii) a quadratic upper time bound for numerical key implication. The applicability of numerical keys to several XML applications is further illustrated in Section 4. In Section 5 we survey the techniques of representing many-to-many relationships within XML, and we conclude in Section 6.

2 Preliminaries

In this section we recall the basics of the XML tree model, the notion of value equality, and describe the path language used to locate sets of nodes within an XML document. Subsequently, we introduce cardinality constraints for XML and show that reasoning about these constraints is computationally intractable. Finally, we identify numerical keys as an important subclass of cardinality constraints.

2.1 The XML Tree Model

XML documents can be modelled as node-labelled trees. We assume that there is a countably infinite set $\mathbf{E}$ denoting element tags, a countably infinite set $\mathbf{A}$ denoting attribute names, and a singleton $\{S\}$ denoting text (PCDATA). We further assume that these sets are pairwise disjoint, and put $\mathcal{L} = \mathbf{E} \cup \mathbf{A} \cup \{S\}$. We refer to the elements of $\mathcal{L}$ as *labels*.

An *XML tree* is a 6-tuple $T = (V, lab, ele, att, val, r)$ where V denotes a set of nodes, and lab is a mapping $V \to \mathcal{L}$ assigning a label to every node in V. A node $v \in V$ is called an *element node* if $lab(v) \in \mathbf{E}$, an *attribute node* if $lab(v) \in \mathbf{A}$, and a *text node* if $lab(v) = S$. Moreover, ele and att are partial mappings defining the edge relation of T: for any node $v \in V$, if v is an element node, then $ele(v)$ is a list of element and text nodes in V and $att(v)$ is a set of attribute nodes in V. If v is an attribute or text node then $ele(v)$ and $att(v)$ are undefined. val is a partial mapping assigning a string to each attribute and text node: for each node $v \in V$, $val(v)$ is a string if v is an attribute or text node, while $val(v)$ is undefined otherwise. Finally, r is the unique and distinguished root node.

An XML tree is said to be *finite* if V is finite, and is said to be *empty* if V consists of the root node only. For a node $v \in V$, each node w in $ele(v)$ or $att(v)$ is called a *child* of v, and we say that there is an edge (v, w) from v to w in T. An XML tree has a tree structure: for each node $v \in V$, there is a unique (directed) path of edges from the root r to v.

We can now define value equality for pairs of nodes in an XML tree. Informally, two nodes u and v of an XML tree T are value equal if they have the same label and, in addition, either they have the same string value if they are text or attribute nodes, or their children are pairwise value equal if they are element nodes. More formally, two nodes $u, v \in V$ are *value equal*, denoted by $u =_v v$, if and only if the subtrees rooted at u and v are isomorphic by an isomorphism that is the identity on string values. That is, two nodes u and v are value equal when the following conditions are satisfied:

(a) $lab(u) = lab(v)$,

(b) if u, v are attribute or text nodes, then $val(u) = val(v)$,

(c) if u, v are element nodes, then *(i)* if $att(u) = \{a_1, \ldots, a_m\}$, then $att(v) = \{a'_1, \ldots, a'_m\}$ and there is a permutation π on $\{1, \ldots, m\}$ such that $a_i =_v a'_{\pi(i)}$ for $i = 1, \ldots, m$, and *(ii)* if $ele(u) = [u_1, \ldots, u_k]$, then $ele(v) = [v_1, \ldots, v_k]$ and $u_i =_v v_i$ for $i = 1, \ldots, k$.

For example, all *teacher*-nodes in Figure 1 are value equal.

2.2 Path Languages

In order to define cardinality constraints we need a path language that is expressive enough to be practical, yet sufficiently simple to be reasoned about efficiently. This is the case for the path languages PL_s and PL (Buneman et al. 2002, Buneman et al. 2003).

Path Language	Syntax
PL_s	$P ::= . \mid \ell \mid P/P$
PL	$Q ::= . \mid \ell \mid Q/Q \mid Q//Q$

Table 1: The path languages PL_s and PL.

A *path expression* is a (possibly empty) finite list of symbols. In this paper, a *simple path expression* is a path expression that consists of labels from $\mathcal{L}$. We use the languages PL_s and PL to describe path expressions. Both languages are XPath fragments, and as usual / denotes parent-child navigation, and // denotes ancestor-descendant navigation.

A path p is a sequence of pairwise distinct nodes $v_0, \ldots, v_m$ where (v_{i-1}, v_i) is an edge for $i = 1, \ldots, m$. We call p a path from v_0 to v_m, and say that v_m is *reachable* from v_0 following the path p. The path p gives rise to a valid simple path expression $lab(v_1)/\cdots/lab(v_m)$, which we denote by $lab(p)$. A path expression is said to be valid if none of its attribute labels occurs at a position different from the last. Let P be a simple path expression, and Q a PL expression. A path p is called a *P-path* if $lab(p) = P$, and a *Q-path* if $lab(p) \in Q$. If p is a path from v to w, then w is said to be reachable from v *following* a P-path or Q-path, respectively. We also write $T \models P(v, w)$ and $T \models Q(v, w)$ when w is reachable from v following a P-path or Q-path, respectively, in an XML tree T. For example, in the XML tree in Figure 1, all *teacher* nodes are reachable from the root following a *year/semester/course/teacher*-path. They are also reachable from the root following a *//teacher*-path.

For a node v of an XML tree T, let $v[\![Q]\!]$ denote the set of nodes in T that are reachable from v by following the PL expression Q, i.e., $v[\![Q]\!] = \{w \mid T \models Q(v, w)\}$. We shall use $[\![Q]\!]$ as an abbreviation for $r[\![Q]\!]$ where r is the root of T. For example, let v be the second *semester* node in Figure 1. Then $v[\![.//\text{student}]\!]$ is the set of all *student* nodes that are children of the second semester. Furthermore, $[\![.//\text{teacher}]\!]$ is the set of all *teacher* nodes in the entire XML tree.

For nodes v and v' of T, the *value intersection* of $v[\![Q]\!]$ and $v'[\![Q]\!]$ is given by $v[\![Q]\!] \cap_v v'[\![Q]\!] = \{(w, w') \mid w \in v[\![Q]\!], w' \in v'[\![Q]\!], w =_v w'\}$ (Buneman et al. 2003). That is, $v[\![Q]\!] \cap_v v'[\![Q]\!]$ consists of all those node pairs in T that are value equal and are reachable from v and v', respectively, by following Q-paths.

A PL expression Q is said to be *contained* in a PL expression Q', denoted by $Q \subseteq Q'$, if for any XML tree T and any node v of T we have $v[\![Q]\!] \subseteq v[\![Q']\!]$. That is, every node that is reachable from v by following a Q-path is also reachable from v by following a Q'-path. Note that $Q \subseteq Q'$ holds if and only if every valid path expression $P \in Q$ satisfies $P \in Q'$ (Buneman et al. 2003). The *containment problem* of PL is to decide, given any PL expressions Q and Q', whether $Q \subseteq Q'$ holds. The containment problem of PL is decidable in $\mathcal{O}(|Q| \times |Q'|)$ time (Buneman et al. 2003).

The choice of a path language is directly influenced by the complexity of its containment problem. It has been argued (Buneman et al. 2002, Buneman et al. 2003) that PL is simple yet expressive enough to be adopted by XML designers and maintained by systems for XML applications.

2.3 Cardinality Constraints and Keys

Let $\mathbb{N}$ denote the positive integers, and $\bar{\mathbb{N}}$ denote the positive integers together with ∞.

Definition 2.1. *A* cardinality constraint *φ for XML is an expression*

$$\text{card}(Q, (Q', \{Q_1, \ldots, Q_k\})) = (\min, \max)$$

where $Q, Q', Q_1, \ldots, Q_k$ are PL expressions such that Q/Q' is a valid path expression if $k = 0$, and $Q/Q'/Q_i$ are valid path expressions for all $i = 1, \ldots, k$ if $k > 0$, where k is a non-negative integer, and where $\min \in \mathbb{N}$ *and* $\max \in \bar{\mathbb{N}}$ *with* $\min \leq \max$. *Herein, Q is called the* context path*, Q' is called the* target path*, $Q_1, \ldots, Q_k$ are called* key paths*,* min *is called the* lower bound*, and* max *the* upper bound *of φ. If $Q = .$, we call φ* absolute*; otherwise φ is called* relative. □

If φ denotes a cardinality constraint, then Q_φ denotes its context path, Q'_φ its target path, $Q^\varphi_1, \ldots, Q^\varphi_{k_\varphi}$ its key paths, $\min_\varphi$ its lower bound and $\max_\varphi$ its upper bound. The *size* $|\varphi|$ of a cardinality constraint φ is defined as the sum of the lengths of all path expressions in φ, i.e., $|\varphi| = |Q_\varphi| + |Q'_\varphi| + \sum_{i=1}^{k_\varphi} |Q^\varphi_i|$.

Let $\sharp S$ denote the cardinality of a finite set S, i.e., the number of its elements.

Definition 2.2. *Let*

$$\varphi = \text{card}(Q, (Q', \{Q_1, \ldots, Q_k\})) = (\min, \max)$$

be a cardinality constraint. An XML tree T satisfies *φ, denoted by $T \models \varphi$, if and only if for all $q \in [\![Q]\!]$, for all $q'_0 \in q[\![Q']\!]$ such that for all $x_1, \ldots, x_k$ with $x_i \in q'_0[\![Q_i]\!]$ for $i = 1, \ldots, k$, the following holds:*

$$\min \leq \sharp\{q' \in q[\![Q']\!] \mid \exists y_1, \ldots, y_k. \forall i = 1, \ldots, k. \; y_i \in q'[\![Q_i]\!] \wedge x_i =_v y_i\} \leq \max. \quad \square$$

To illustrate the definitions above, we formalise the constraints from the introduction. Let T be the XML tree in Figure 1. The constraint

$$\text{card}(.//\text{semester}, (\text{course}, \{.//\text{sid}\})) = (2, 4)$$

says that in every semester each student who decides to study must enrol in at least two and at most four courses. T violates this constraint since *Bart Simpson* is only enrolled in one course in the second semester of 2007. However, T does satisfy the second constraint

$$\text{card}(\text{year}, (\text{semester}, \emptyset)) = (2, 2)$$

that states that each *year*-node has either precisely two *semester*-children or none at all. The cardinality constraint

$$\text{card}(\text{year}, (.//\text{course}, \{\text{teacher}\})) = (3, 6)$$

is satisfied while

$$\text{card}(.//\text{semester}, (\text{course}, \{\text{teacher}\})) = (3, 3)$$

is violated by T since *Principal Skinner* only teaches *maths* and *physics* within semester *1* of *2007*. An example of an absolute cardinality constraint is

$$\text{card}(., (.//\text{course}, \{\text{name}, \text{student/sid}\})) = (1, 3),$$

i.e., a student may attempt to pass the same course up to 3 times. XML keys as introduced by Buneman et al. (Buneman et al. 2002) and further studied in (Buneman et al. 2003, Hartmann & Link 2007*b*) are completely covered by numerical constraints. More specifically, the cardinality constraint φ is an XML key precisely when $min_\varphi = max_\varphi = 1$. Major observations that hold for the definition of XML keys (Buneman et al. 2002, Hartmann & Link 2007*b*) therefore also apply to cardinality constraints. Some examples of XML keys are $\text{card}(., (\text{year}, \{\text{calendar}\}) = (1, 1)$ stating that *year*-nodes can be identified by the value of their *calendar*-child, $\text{card}(\text{year}, (\text{semester}, \{\text{no}\}) = (1, 1)$ stating that *semester*-nodes can be identified by their *no*-child relatively to the *year*, $\text{card}(.//\text{semester}, (\text{course}, \{\text{name}\}) = (1, 1)$ stating that *course*-nodes can be identified by their *name*-child relatively to the *semester*, and $\text{card}(.//\text{course}, (\text{student}, \{\text{sid}\}) = (1, 1)$ stating that *student*-nodes can be identified by their *sid*-child relatively to the *course*.

2.4 Satisfiability and Implication

In this section we define fundamental decision problems associated with various classes of constraints. Let Σ be a finite set of constraints in a class $\mathcal{C}$ and T be an XML tree. We say that T *satisfies* Σ if and only if $T \models \sigma$ for every $\sigma \in \Sigma$. The *(finite) satisfiability problem* for $\mathcal{C}$ is to determine, given any finite set Σ of constraints in $\mathcal{C}$, whether there is a (finite) XML tree satisfying Σ.

Let $\Sigma \cup \{\varphi\}$ be a finite set of constraints in $\mathcal{C}$. We say that Σ *(finitely) implies* φ, denoted by $\Sigma \models_{(f)} \varphi$, if and only if every (finite) XML tree T that satisfies Σ also satisfies φ. The *(finite) implication problem* is to decide, given any finite set of constraints $\Sigma \cup \{\varphi\}$, whether $\Sigma \models_{(f)} \varphi$. If Σ is a finite set of constraints in $\mathcal{C}$ let Σ^* denote its *semantic closure*, i.e., the set of all constraints implied by Σ. That is, $\Sigma^* = \{\varphi \in \mathcal{C} \mid \Sigma \models \varphi\}$.

2.5 Computational Intractability

For XML keys, the finite and the unrestricted implication problem coincide (Buneman et al. 2003). For numerical constraints, however, the situation is already different for the satisfiability problem.

Theorem 2.1. *The satisfiability problem and the finite satisfiability problem for the class of cardinality constraints do not coincide.* □

Still, satisfiability for cardinality constraints can be decided easily. This is due to the fact that the empty XML tree satisfies every cardinality constraint that contains at least one label from $\mathcal{L}$. Deciding implication, on the other hand, is not that easy. We now demonstrate that reasoning is likely to be computationally intractable already for very restricted classes of cardinality constraints. We call a cardinality constraint $\mathrm{card}(P, P', \{P_1, \ldots, P_k\}) = (\min, \max)$ *simple* if $P, P', P_1, \ldots, P_k$ are all simple path expressions in PL_s.

Theorem 2.2. *The finite implication problem for the class of all simple absolute cardinality constraints with a non-empty set of key paths is coNP-hard.* □

The previous result suggests that computational intractability may result from the specification of both lower and upper bounds. The next result indicates that an empty set of key paths may cause computational intractability.

Theorem 2.3. *The finite implication problem for the class of all simple absolute cardinality constraints where the lower bound is fixed to 1 is coNP-hard.* □

The next theorem suggests another source of computational intractability: the permission to have arbitrary path expressions in both target- and key paths.

Theorem 2.4. *The finite implication problem for the class of all absolute cardinality constraints that have a non-empty set of key paths and where the lower bound is fixed to 1 is coNP-hard.* □

2.6 Numerical Keys

In order to take advantage of XML applications effectively it becomes necessary to reason about constraints efficiently. The results from the last section motivate the study of a large subclass of cardinality constraints that, as we shall show in this paper, turns out to be computationally tractable.

Definition 2.3. *A* numerical key *for XML is a cardinality constraint*

$$\mathrm{card}(Q, (Q', \{P_1, \ldots, P_k\})) = (1, \max)$$

where k is a positive integer and $P_1, \ldots, P_k$ are simple path expressions in PL_s. *We will use* $\mathrm{card}(Q, (Q', \{P_1, \ldots, P_k\})) \leq \max$ *to denote numerical keys. Let $\mathcal{N}$ denote the class of all numerical keys.* □

Notice that numerical keys are still much more expressive than XML keys. More specifically, a numerical key φ becomes a key precisely when $\max_\varphi = 1$. The next result states that every finite set of numerical keys can be satisfied by some finite XML tree: we can just choose the empty XML tree. This extends a result for XML keys (Buneman et al. 2003).

Theorem 2.5. *Every finite set of numerical keys is finitely satisfiable.* □

Moreover, the coincidence of finite and unrestricted implication carries over from keys to numerical keys. In what follows we will therefore speak of the implication problem for numerical keys.

Theorem 2.6. *The implication and finite implication problems for numerical keys coincide.* □

3 Reasoning about Numerical Keys

The notion of derivability ($\vdash_{\mathfrak{R}}$) with respect to a set $\mathfrak{R}$ of inference rules can be defined analogously to the notion in the relational data model (Abiteboul, Hull & Vianu 1995, pp. 164-168). In the first part of this section we will find a set $\mathfrak{R}$ of inference rules which is *sound*, i.e. $\Sigma^+_{\mathfrak{R}} \subseteq \Sigma^*$, and *complete*, i.e. $\Sigma^* \subseteq \Sigma^+_{\mathfrak{R}}$, for the implication of numerical keys, and where $\Sigma^+_{\mathfrak{R}} = \{\varphi \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$ denotes the *syntactic closure* of Σ under inference using $\mathfrak{R}$. In the second part of this section we will provide an upper bound for the time-complexity of deciding numerical key implication.

3.1 Axiomatisation

Table 2 shows a set of inference rules for the implication of numerical keys. The majority of these rules are extensions of the inference rules for keys (Buneman et al. 2003, Hartmann & Link 2007*b*). Moreover, *infinity*, *weakening* and *superkey* rule form an axiomatisation for max-cardinality constraints in the Entity-Relationship model (Hartmann 2001).

It is not difficult to show the soundness of the inference rules in Table 2 for the implication of numerical keys. The completeness argument uses shortest path methods (Hartmann & Link 2007*a*).

Theorem 3.1. *The inference rules from Table 2 are sound and complete for the implication of numerical keys in $\mathcal{N}$.*

3.2 Time-complexity of Deciding Implication

The technique of proving completeness can be extended to obtain an algorithm for deciding numerical key implication in time quadratic in the size of the constraints (Hartmann & Link 2007*a*).

Theorem 3.2. *Let $\Sigma \cup \{\varphi\}$ be a finite set of numerical keys in $\mathcal{N}$. The implication problem $\Sigma \models \varphi$ can be decided in time $\mathcal{O}(|\varphi| \cdot (||\Sigma|| + |\varphi|))$.* □

$$\frac{}{card(Q,(Q',S)) \leq \infty}$$ (infinity)

$$\frac{}{card(Q,(.,S)) \leq 1}$$ (self)

$$\frac{card(Q,(Q',S)) \leq max}{card(Q,(Q',S)) \leq max+1}$$ (weakening)

$$\frac{card(Q,(Q',S)) \leq max}{card(Q,(Q',S \cup \{P\})) \leq max}$$ (superkey)

$$\frac{card(Q,(Q'/P,\{P'\})) \leq max}{card(Q,(Q',\{P/P'\})) \leq max}$$ (subnodes)

$$\frac{card(Q,(Q'.Q'',S)) \leq max}{card(Q.Q',(Q'',S)) \leq max}$$ (target-to-context)

$$\frac{card(Q,(Q',S)) \leq max}{card(Q'',(Q',S)) \leq max}\, Q'' \subseteq Q$$ (context-path-containment)

$$\frac{card(Q,(Q'/P,\{.,P'\})) \leq max}{card(Q,(Q',\{.,P/P'\})) \leq max}$$ (subnodes-self)

$$\frac{card(Q,(Q',S)) \leq max}{card(Q,(Q'',S)) \leq max}\, Q'' \subseteq Q'$$ (target-path-containment)

$$\frac{card(Q,(Q',S \cup \{.,P\})) \leq max}{card(Q,(Q',S \cup \{.,P/P'\})) \leq max}$$ (prefix-self)

$$\frac{\begin{array}{c} card(Q,(Q',\{P/P_1,\ldots,P/P_k\})) \leq max, \\ card(Q/Q',(P,\{P_1,\ldots,P_k\})) \leq max' \end{array}}{card(Q,(Q'/P,\{P_1,\ldots,P_k\})) \leq max \cdot max'}$$ (multiplication)

Table 2: An Axiomatisation for Numerical Keys

4 Applications

The introduction has already revealed that several XML recommendations would benefit from the specification of numerical keys. In particular, we have seen examples for predicting the number of query answers for XPath and XQuery queries in the presence of numerical keys. In the same way one is able to make predictions for the number of updates, using for instance the XQuery update facility. For example, the XQuery query

for $s **in**
  doc("enrol.xml")/year[@calendar="2007"]/
  semester[@no="2"]//student[@sid="247"]
return do replace value of $s/last
  with "Milhouse"

will update the lastname of the student with @*sid*=247 by *Milhouse* in all course enrolments of this student in semester two of 2007. If the database management system is able to conclude that

$$\text{card}(.//\text{semester}, (.//\text{student}, \{\text{sid}\})) \leq 4$$

is implied by the set Σ of numerical keys specified by the database designer, then the maximal number of updates this query causes is 4.

When XML data is exchanged over the Web it is very common that sensitive information is encrypted, e.g. by XML encryption. In order to evaluate queries on encrypted XML data it may become necessary to decrypt certain data element nodes in order to return the relevant information in the answer. If we recall the example of XQuery query 1 from the introduction and assume that grade elements have been encrypted, then a database management system capable of inferring that both

$$\text{card}(\text{year}, (.//\text{student}, \{\text{sid}\})) \leq 8$$

and

$$\text{card}(.//\text{student}, (\text{grade}, \emptyset)) \leq 1$$

are implied by the constraints specified can also predict that the number of necessary decryptions to answer this query is at most 8, and thus also predict the time necessary to deliver the information requested.

As a last application we demonstrate the potential of cardinality constraints for generating XML views to efficiently process common types of queries and updates. Recall the cardinality constraint that each

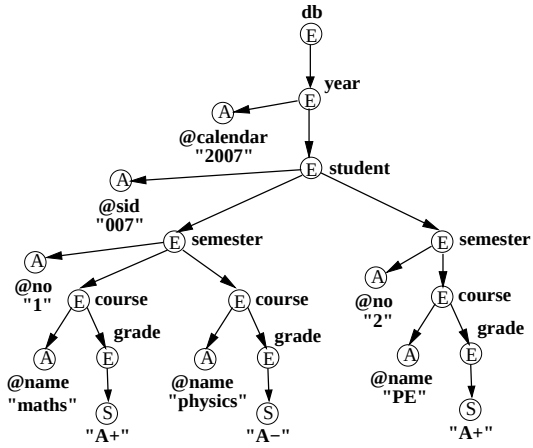

Figure 2: XML View Fragment

year-node subsumes up to eight *course*-nodes that contain *student*/*@sid*-subnodes with the same value. It could be rather expensive to query the original XML tree for course information based on a specific *year* and a specific *sid* since for all *course*-nodes (in the worst case) it has to be decided whether they have an *sid*-subnode with that particular value. Querying becomes even more inefficient when some of this information is encrypted.

Instead, one may create the XML view in Figure 2 (showing the fragment for *Bart Simpson* only), rewrite query (1) and use the XML view to evaluate the resulting query.

On the XML view the constraint from before translates into the following occurrence constraint: under each *student*-node there are up to eight *course*-nodes. Since students can be identified by their sid relatively to the year in this XML view it is therefore better to pose queries on course information based on a specific year and a specific sid against this XML view. Then query (1) is rewritten into the following query:

for $c **in** doc("view.xml")/
    year[@cal="2007"]/student[@sid="007"]/course
return ⟨grade⟩{$c/grade}⟨/grade⟩

The selection of *student*-elements based on their *sid* in early location steps achieves a better performance. Thus, the creation of XML views based on cardinality

constraints may lead to simplified integrity checking, and more efficient processing of common queries and updates.

5 Many-to-Many Relationships in XML

There have been many proposals for transforming data from various data models, such as the Relational Data Model (RDM) (Lee, Mani, Chiu & Chu 2002, Duta, Barker & Alhajj 2004, Liu, Liu & Guo 2003, Lv & Yan 2006, Wang, Lo, Alhajj & Barker 2005, Vincent, Liu & Liu 2003, Liu, Vincent & Liu 2006), Entity Relationship Model (ERM) (Kleiner & Lipeck 2001, Liu & Li 2006, Pigozzo & Quintarelli 2005) and Unified Modelling Language (UML) (Kudrass & Krumbein 2003) to XML. On the one hand such mappings can facilitate the extraction of existing data for data exchange. On the other, existing conceptual models can also be used for modelling XML.

The translation of 1-to-1 and 1-to-many relationships into a hierarchical model is relatively straightforward. The real challenges reveal themselves with many-to-many relationships. Naturally, there are some important integrity constraints which we would want the transformation to preserve: two common classes of ER constraints are key constraints and participation cardinality constraints. In the following, we present a collection of mappings of a many-to-many relationship type into XML Schema, the defacto standard for XML. We start with mappings which appear relatively straightforward and intuitive. However, it turns out that they do not necessarily preserve all key constraints and participation constraints. Therefore, we try to enhance these mappings in order to preserve more and more constraints.

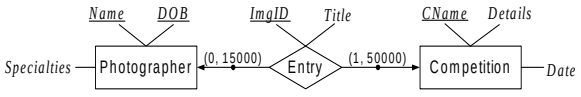

Figure 3: Example of a many-to-many relationship.

For demonstrating the various transformation approaches, let us consider a concrete many-to-many ER relationship which models entries into photographic competitions. A photographer may submit many entries into competitions and a competition certainly may receive many entries. The ER diagram depicting this many-to-many relationship is shown in Figure 3. The entity type `Photographer` has key {Name,DOB}, and the relationship type `Entry` has key attribute *ImgID* as well as both of its roles as key components, that is `Photographer` and `Competition`. Furthermore, each photographer enters up to 15000 entries while each competition must receive at least one entry but up to 50000 entries.

5.1 Representing XML Schema

We will represent the main features of an XML schema definition with the help of XML schema trees. Furthermore, XML constraints other than occurrence constraints are specified explicitly. An XML tree with $val(v)$ being defined for at least one node v is called a *data tree*. A *schema tree* is an XML tree together with occurrence constraints between parent-child elements as defined by *ele*, where no two siblings have the same name and kind. In particular, $val(v)$ is undefined for all nodes $v \in V$.

An element is said to *represent* a role (or relationship type) if an XML key with this element as its target is required for preserving the key constraint for the corresponding role (or relationship type). That is, there is a bijection between the elements representing some role (or relationship type) and the entity set (or relationship set) for such role (or relationship type). We distinguish such an element by type $\mathbf{E}^r$.

We denote an XML Schema key constraint by a statement of the following form:

$$\mathcal{K}_{name}: \text{KEY}(context_p, (target_p, \{p_1, \ldots, p_n\}))$$

where $context_p$, $target_p$ and p_i are all simple path expressions such that each $context_p/target_p/p_i$ is a valid path expression with respect to some given XML schema graph. A path expression is *valid* with respect to a schema graph T if the expression describes a dipath in T. The XML key expression above corresponds to the following XML Schema key specification defined in the scope of the last element in $context_p$:

⟨xs:key name="$\mathcal{K}_{name}$"⟩
  ⟨xs:selector xpath="$target_p$"/⟩
  ⟨xs:field xpath="p_1"/⟩
...
  ⟨xs:field xpath="p_n"/⟩
⟨/xs:keyref⟩

An XML Schema keyref constraint is expressed by an expression of the form:

$$\mathcal{F}_{name}: (context_q, (target_q, \{q_1, \ldots, q_n\})) \text{ REFERENCES } \mathcal{K}_{name}$$

where every $context_q/target_q/q_i$ is a valid path expression with respect to the schema graph on which $\mathcal{K}_{name}$ is defined and, the context path of $\mathcal{K}_{name}$ is a prefix path of $context_q$. This foreign key expression corresponds to the following XML Schema keyref specification defined in the scope of the last element in $context_q$:

⟨xs:keyref name="$\mathcal{F}_{name}$" refer="$\mathcal{K}_{name}$"⟩
  ⟨xs:selector xpath="$target_q$"/⟩
  ⟨xs:field xpath="q_1"/⟩
...
  ⟨xs:field xpath="q_n"/⟩
⟨/xs:keyref⟩

For convenience, we will abbreviate every label to its first two letters. For example, the label "Photographer" is abbreviated to `Ph`.

5.2 Mapping Approaches

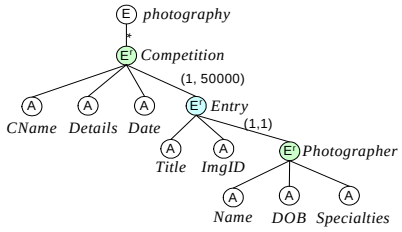

Figure 4: Example of the Nested Approach, with base `Competition`.

Nested Approach. For our first mapping attempt, we simply mimic how the entities and relationship are connected in the ER diagram. We choose one role of the relationship as a base for the hierarchy, that is, a root for the sub-hierarchy to be nested

under a unique root. For example, if we choose the base `Competition` then we obtain the XML schema graph shown in Figure 4, where the root of the subhierarchy is the `Competition` element which has a child element `Entry`, which in turn has a child element `Photographer`. The key constraints for `Competition` and `Entry` can be preserved by adding the following XML schema keys respectively:

$$\mathcal{K}_{\texttt{Competition}}: \text{KEY}(., (\texttt{Co}, \{@\texttt{CN}\}))$$
$$\mathcal{K}_{\texttt{Entry}}: \text{KEY}(\texttt{Co}, (\texttt{En}, \{@\texttt{Im}, \texttt{Ph}/@\texttt{Na}, \texttt{Ph}/@\texttt{DO}\}))$$

Moreover, this means that the cardinality constraint `card(Entry,Competition)`=(1, 50000) is preserved by an occurrence constraint between the `Competition` and `Entry` elements. Note that the occurrence constraint preserves `card(Entry,Competition)`=(1, 50000) as expected, in part because of $\mathcal{K}_{\texttt{Entry}}$.

Figure 5 illustrates an XML data tree for an XML document which is valid with respect to the photography XML schema in Figure 4.

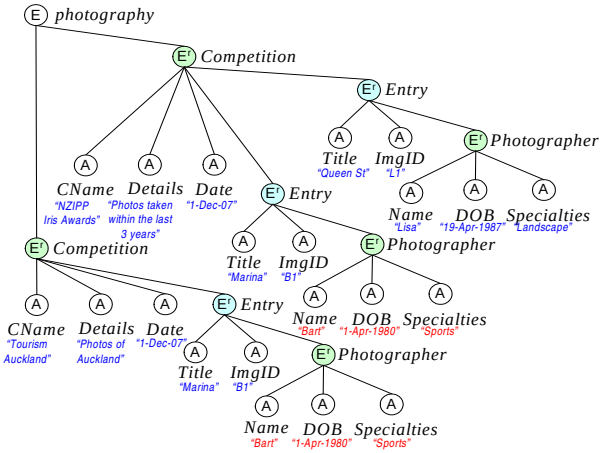

Figure 5: Sample XML data tree for Nested Approach with base `Competition`.

The nested approach has three problems: i) the corresponding data mapping may not be lossless, ii) we cannot preserve the key constraint for a non-base role and, iii) we cannot preserve the cardinality constraint relating to a non-base role. The data loss problem is encountered if there is a photographer who has not entered into any competition, which is clearly possible. For this specific example, we can avoid data loss by simply choosing `Photographer` as the base. However this solution does not work in general, specifically it fails when the cardinality constraints for both roles have a lower bound of 0. Since a photographer may submit many entries, we can only specify relative XML keys whose target path ends in `Photographer`. As demonstrated by the data tree, a relative XML key for `Photographer` does not identify a photographer in the same way as the corresponding ER key. We cannot express that photographer Bart from the NZIPP Iris Awards competition is in fact the same person as the photographer Bart from the Tourism Auckland competition. With this approach we would struggle with designing updateable XML views as well as having to cope with data redundancy.

Inline Approach. To solve the identification problem, we can treat the two roles in the same manner. That is, we nest the elements representing each role directly under the unique root. The child element of the relationship element is instead just a reference

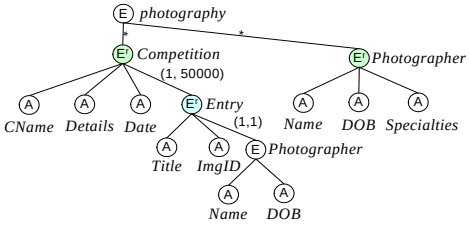

Figure 6: Example of the Inline Approach.

to the element representing the non-base role. For our example, the approach yields the XML schema graph in Figure 6. the two XML keys from the Nested Approach and, the following XML constraints:

$$\mathcal{K}_{\texttt{Photographer}}: \text{KEY}(., (\texttt{Ph}, \{@\texttt{Na}, @\texttt{DO}\}))$$
$$\mathcal{F}^{\texttt{Entry}}_{\texttt{Photographer}}: (., (\texttt{Co}/\texttt{En}/\texttt{Ph}, \{@\texttt{Na}, @\texttt{DO}\}))$$
$$\text{REFERENCES } \mathcal{K}_{\texttt{Photographer}}$$

The Inline Approach overcomes the first two problems of the Nested Approach, but we are still not able to preserve the cardinality constraint of the non-base role. Using references to represent the association between roles and a relationship type is one of the most common mapping approach in the literature.

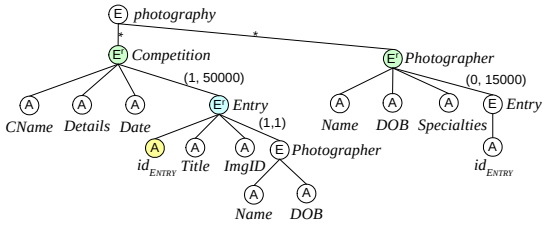

Figure 7: Example of the Inline-With-Identifier Approach.

Inline-With-Identifier Approach. XML Schema only provides occurrence constraints as a means to express the number of element children. Therefore, in order to preserve a second cardinality constraint, we first need to replicate information about the relationship under the non-base role. One way to do this is to introduce a surrogate identifying attribute for the element representing the relationship. In this way the non-base role can simply contain a reference to the relationship element via this surrogate identifier. Under the Inline-With-Identifier Approach we obtain the XML schema graph in Figure 7, the four XML constraints from the Inline Approach and the following:

$${}^{i}\mathcal{K}_{\texttt{Entry}}: \text{KEY}(., (\texttt{Co}/\texttt{En}, \{@id_{\text{ENTRY}}\}))$$
$${}^{i}\mathcal{F}^{\texttt{Photographer}}_{\texttt{Entry}}: (., (\texttt{Ph}/\texttt{En}, \{@id_{\text{ENTRY}}\}))$$
$$\text{REFERENCES } {}^{i}\mathcal{K}_{\texttt{Entry}}$$
$${}^{i}\mathcal{K}_{\texttt{Ph}/\texttt{Entry}}: \text{KEY}(., (\texttt{Ph}/\texttt{En}, \{@id_{\text{ENTRY}}\}))$$
$$\mathcal{F}^{\texttt{Entry}}_{\texttt{Photographer}}: (., (\texttt{Co}/\texttt{En}, \{@id_{\text{ENTRY}}\}))$$
$$\text{REFERENCES } {}^{i}\mathcal{K}_{\texttt{Ph}/\texttt{Entry}}$$

These four constraints present a first step towards ensuring that we are actually replicating relationship

data consistently. In particular, replicated Entry elements refer only to an element representing some actual competition entry and, every element representing an Entry relationship must have exactly one participating Photographer entity. The second constraint is implicit in the ER model.

There is still the problem of ensuring that an element representing the Entry entity type and the one which references such an element are connected to the same photographer. In other words, for every Entry element nested under some Photographer element, the Photographer element must be the one referenced by the Entry element that is nested under some Competition element and sharing the same $@id_{\text{ENTRY}}$ value. Since there may be many Entry children elements for some Photographer element, keyrefs cannot be used. As in the relational data model we may specify a more general inclusion constraint such as:

$$\mathcal{I}^{\texttt{Photographer}}_{\texttt{Entry}} : (., (\texttt{Ph}, \{@\texttt{Na}, @\texttt{DO}, \texttt{En}/@id_{\text{ENTRY}}\})) \sqsubseteq (., (\texttt{Co}/\texttt{En}, \{\texttt{Ph}/@\texttt{Na}, \texttt{Ph}/@\texttt{DO}, @id_{\text{ENTRY}}\}))$$

The inclusion constraint states that, in the data tree, for every element $\langle p \rangle$ representing some Photographer entity, there exist elements $\langle e_1 \rangle, \ldots, \langle e_n \rangle$ representing an Entry relationship such that the following holds. The set of valuations $\mathcal{I}_p$ for the set of paths on the left-hand side of the inclusion constraint is included in the set of valuations $\mathcal{I}_e$ for the set of paths on the right-hand side of the inclusion constraint. Here, $\mathcal{I}_p$ and $\mathcal{I}_e$ consist of those valuations whose paths are rooted at $\langle p \rangle$ and $\langle e_1 \rangle, \ldots, \langle e_n \rangle$, respectively.

A consequence of not specifying $\mathcal{I}^{\texttt{Photographer}}_{\texttt{Entry}}$ (or its equivalent constraint), is that card(Entry,Photographer)=(0, 15000) is not preserved by an occurrence constraint as we would expect. In addition, there are XML instances which validate against the XML schema definition but do not correspond to any ER instance. This is especially undesirable if we want to use the schema for data exchange and creating updateable views.

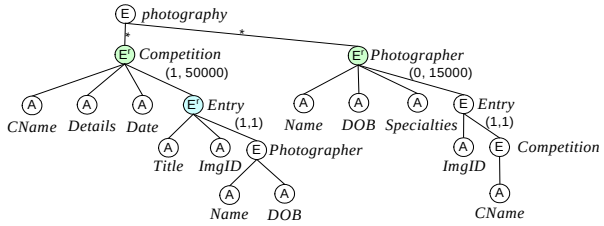

Figure 8: Example of the Roles-As-Parents Approach.

Roles-As-Parents Approach. Rather than introducing a surrogate attribute, this approach replicates relationship data in accordance with the key for the relationship type. Therefore, since the key for Entry includes both roles, the element representing Photographer will have a child Entry element which has a Competition child element reference to the element representing Competition. The structure of the resulting XML schema under the Roles-As-Parents Approach is illustrated in Figure 8. The XML data tree in Figure 9 demonstrates the intended data mapping for this approach. Clearly we need all the XML constraints from the Inline Approach together with a new keyref constraint

$$\mathcal{F}^{\texttt{Photographer}}_{\texttt{Competition}} : (., (\texttt{Ph}/\texttt{En}/\texttt{Co}, \{@\texttt{CN}\})) \text{ REFERENCES } \mathcal{K}_{\texttt{Competition}}$$

that checks that the second Competition actually replicates competition identifiers. Once again, we are faced with the problem of ensuring that the replicated relationship data is consistent with identifiable relationships. For the same reason as above, keyrefs are inadequate and we need more general inclusion constraints:

$$\mathcal{I}^{\texttt{Entry}}_{\texttt{Ph/Entry}} : (., (\texttt{Co}, \{@\texttt{CN}, \texttt{En}/@\texttt{Im}, \texttt{En}/\texttt{Ph}/@\texttt{Na}, \texttt{En}/\texttt{Ph}/@\texttt{DO}\})) \sqsubseteq (., (\texttt{Ph}, \{\texttt{En}/\texttt{Co}/@\texttt{CN}, \texttt{En}/@\texttt{Im}, @\texttt{Na}, @\texttt{DO}\}))$$

$$\mathcal{I}^{\texttt{Ph/Entry}}_{\texttt{Entry}} : (., (\texttt{Ph}, \{\texttt{En}/\texttt{Co}/@\texttt{CN}, \texttt{En}/@\texttt{Im}, @\texttt{Na}, @\texttt{DO}\})) \sqsubseteq (., (\texttt{Co}, \{@\texttt{CN}, \texttt{En}/@\texttt{Im}, \texttt{En}/\texttt{Ph}/@\texttt{Na}, \texttt{En}/\texttt{Ph}/@\texttt{DO}\}))$$

The purpose of the inclusion constraints is to guarantee that every Entry element in the sub-hierarchy with base Competition corresponds to precisely one Entry element in the sub-hierarchy with base Photographer, and vice versa. For example, since the "NZIPP Iris award" has an entry by Bart for the image with ID "B1", we should find for Photographer Bart an entry of image "B1" in the competition "NZIPP Iris award". Observe that we are storing identifying aspects of each relationship twice. As in the previous approach the cardinality constraint card(Entry,Photographer)=(0, 15000) may not be preserved without specifying the inclusion constraints despite having specified an occurrence constraint to achieve just that.

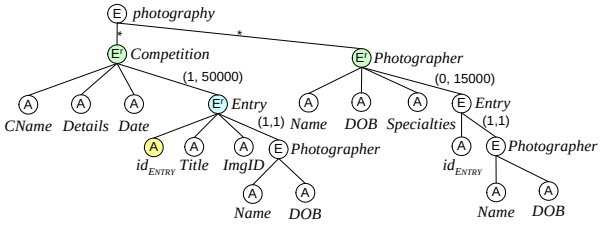

Figure 10: Example of the Self-Replication Approach.

Self-Replication Approach. This approach extends the Inline-With-Identifier Approach such that inclusion constraints can be avoided. We add a child Photographer element to the Entry element that is itself nested under the element representing the Photographer entity type. Figure 10 shows the photography XML schema graph under the Self-Replication Approach. All XML constraints from the Inline-With-Identifier Approach except $\mathcal{I}^{\texttt{Photographer}}_{\texttt{Entry}}$ are needed. We also add the following constraints:

$${}^{s}\mathcal{K}_{\texttt{Photographer}} : \text{KEY}(\texttt{Ph}, (., \{@\texttt{Na}, @\texttt{DO}\}))$$

$${}^{s}\mathcal{F}_{\texttt{Photographer}} : (\texttt{Ph}, (\texttt{En}/\texttt{Ph}, \{@\texttt{Na}, @\texttt{DO}\})) \text{ REFERENCES } {}^{s}\mathcal{K}_{\texttt{Photographer}}$$

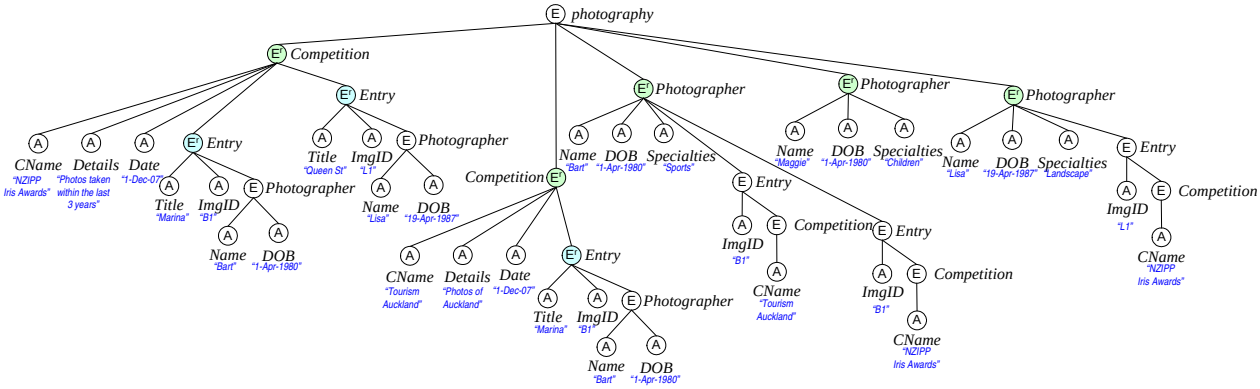

Figure 9: Data tree for the Roles-As-Parents Approach.

and

$$
\begin{aligned}
{}^{i}\mathcal{K}'_{\texttt{Entry}}: \ & \text{KEY}(., (\texttt{Co/En}, \{@id_{\text{ENTRY}}, \\
& \texttt{Ph/@Na}, \texttt{Ph/@DO}\})) \\
{}^{i}\mathcal{F}^{\texttt{Ph/Entry}}_{\texttt{Entry}}: \ & (., (\texttt{Ph/En}, \{@id_{\text{ENTRY}}, \\
& \texttt{Ph/@Na}, \texttt{Ph/@DO}\})) \\
& \text{REFERENCES } \mathcal{K}'_{\texttt{Entry}}
\end{aligned}
$$

In fact ${}^{s}\mathcal{K}_{\texttt{Photographer}}$ is a trivial key, but is necessary in order for us to specify ${}^{s}\mathcal{F}_{\texttt{Photographer}}$ that ensures that the correct photographer identification is replicated for the referencing `Entry` element. We can then use the second key-keyref pair to ensure that the relationship data is replicated consistently. As a result, the occurrence constraint between the element representing `Photographer` and its child `Entry` element would indeed preserve `card`(`Entry`,`Photographer`)=(0, 15000). Although the approach preserves all key constraints and cardinality constraints of a many-to-many relationship, the data structure is not natural and neither are some of the required XML constraints.

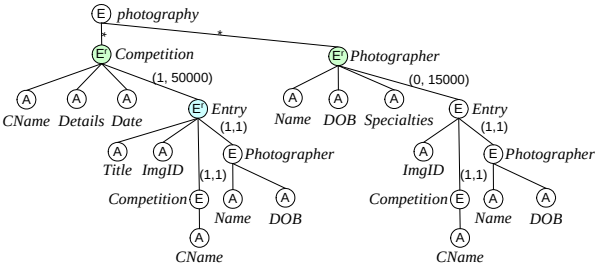

Figure 11: Example of the Relationship-Replication Approach.

Relationship-Replication Approach. To avoid using inclusion constraints without adding a surrogate identifier for the relationship, we can replicate the entire key for the relationship. In particular, each `Entry` element in the XML schema will contain both a `Competition` and a `Photographer` child element since the key for `Entry` includes both roles. This approach can be seen as an extension of the Roles-As-Parents Approach. The photography XML schema structure under this approach is shown in Figure 11. We need every XML constraints from the Roles-As-Parents Approach except for the two inclusion constraints, ${}^{s}\mathcal{K}_{\texttt{Photographer}}$, ${}^{s}\mathcal{F}_{\texttt{Photographer}}$ from the previous mapping and, also the following:

$$
\begin{aligned}
{}^{s}\mathcal{K}_{\texttt{Competition}}: \ & \text{KEY}(\texttt{Co}, (., \{@\texttt{CN}\})) \\
{}^{s}\mathcal{F}_{\texttt{Competition}}: \ & (\texttt{Co}, (\texttt{En/Co}, \{@\texttt{Na}, @\texttt{DO}\})) \\
& \text{REFERENCES } {}^{s}\mathcal{K}_{\texttt{Competition}} \\
{}^{i}\mathcal{K}''_{\texttt{Entry}}: \ & \text{KEY}(., (\texttt{Co/En}, \{@\texttt{Im}, \\
& \texttt{Ph/@Na}, \texttt{Ph/@DO}, \texttt{Co/@CN}\})) \\
{}^{i}\mathcal{F}^{\texttt{Ph/Entry}}_{\texttt{Co/Entry}}: \ & (., (\texttt{Ph/En}, \{@\texttt{Im}, \texttt{Ph/@Na}, \texttt{Ph/@DO}, \\
& \texttt{Co/@CN}\})) \text{ REFERENCES } \mathcal{K}''_{\texttt{Entry}} \\
{}^{i}\mathcal{K}''_{\texttt{Ph/Entry}}: \ & \text{KEY}(., (\texttt{Ph/En}, \{@\texttt{Im}, \\
& \texttt{Ph/@Na}, \texttt{Ph/@DO}, \texttt{Co/@CN}\})) \\
{}^{i}\mathcal{F}^{\texttt{Co/Entry}}_{\texttt{Ph/Entry}}: \ & (., (\texttt{Co/En}, \{@\texttt{Im}, \texttt{Ph/@Na}, \texttt{Ph/@DO}, \\
& \texttt{Co/@CN}\})) \text{ REFERENCES } \mathcal{K}''_{\texttt{Ph/Entry}}
\end{aligned}
$$

Similar to the previous approach, ${}^{s}\mathcal{K}_{\texttt{Competition}}$ is a trivial key but necessary for specifying ${}^{s}\mathcal{F}_{\texttt{Competition}}$. We are then in a position to ensure consistency of relationship replication using just keys and keyrefs. The approach preserves the desired constraints but at the cost of high data replications. Although not data redundancy in the traditional sense, these data replications can still lead to difficulties when dealing with updates.

Inline Approach with Numerical Keys. Recall that the Inline Approach resulted in relatively little data replication whilst failing to preserve only one of the cardinality constraints. Our previous approaches have tried to overcome this problem with just occurrence constraints. The results have been quite intricate XML schema definitions. Occurrence constraints can be used to preserve participation cardinality constraints but not necessarily on their own. So: Is there a more direct way to preserve a cardinality constraint apart from using occurrence constraints? An answer lies, in part, with cardinality constraints for XML. For example, to preserve `card`(`Entry`,`Photographer`)=(0, 15000) we simply need to add the numerical key:

$$\text{CARD}(., (\texttt{Co/En}, \{\texttt{Ph/@Na}, \texttt{Ph/@DO}\})) = (1, 15000)$$

to the Inline Approach. The constraint states that for every distinct valuation of photographer's name and date of birth, there can be between 0 and 15,000 entry elements with a photographer child element having such a valuation for its attributes. Because every `Photographer` child element of some `Entry` element

refers to a photographer, the numerical key preserves the cardinality constraint relating to photographers.

However, numerical keys alone do not suffice in the case that the cardinality constraint we want to preserve has a lower bound greater than 0. In this case some form of existence constraint is required. Suppose we only consider that every photographer has between 1 and 15,000 entries into competitions. Then, in addition to a numerical key, we need a way to say that every `Photographer` element that is nested under the unique root must be referenced by at least one `Photographer` element which is nested under some `Entry` element.

6 Conclusion

We have briefly summarised the benefits of introducing cardinality constraints to XML. They can express many important semantic information about XML data that cannot be expressed by common XML schema languages. Moreover, we have illustrated that many XML applications can benefit from the specification of such constraints. While reasoning about cardinality constraints is intractable in general we have identified the large subclass of numerical keys that is finitely satisfiable, finitely axiomatisable, and whose implication problem can be decided in quadratic time. We have established that the implication problem of numerical keys can be characterised as a shortest path problem in a certain digraph. Thus, numerical keys form a very natural and robust class of XML constraints that can be utilised effectively by designers, and the complexity of their associated decision problems indicates that they can be maintained efficiently by database systems for XML applications.

Moreover, we have demonstrated the difficulty of representing many-to-many relationships adequately in XML. Since the semantics of these relationships is often modelled with the help of key, foreign key and participation constraints, these must be represented in XML appropriately as well. We have shown a variety of approaches and discussed their advantages and disadvantages.

References

Abiteboul, S., Hull, R. & Vianu, V. (1995), *Foundations of Databases*, Addison-Wesley.

Apparao et al, V. (n.d.), 'Document object model (DOM) Level 1 Specification, W3C Recommendation, Oct. 1998', http://www.w3.org/TR/REC-DOM-Level-1/.

Arenas, M. & Libkin, L. (2004), 'A normal form for XML documents', *TODS* **29**(1), 195–232.

Arenas, M. & Libkin, L. (2005), 'An information-theoretic approach to normal forms for relational and XML data', *J. ACM* **52**(2), 246–283.

Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J. & Siméon, J. (n.d.), 'XQuery 1.0: An XML query language, W3C Recommendation, Jan. 2007', http://www.w3.org/TR/xquery/.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. & Yergeau, F. (n.d.), 'Extensible markup language (XML) 1.0 (Fourth Edition) W3C Recommendation, Aug. 2006', http://www.w3.org/TR/xml/.

Buneman, P., Davidson, S., Fan, W., Hara, C. & Tan, W. (2002), 'Keys for XML', *Computer Networks* **39**(5), 473–487.

Buneman, P., Davidson, S., Fan, W., Hara, C. & Tan, W. (2003), 'Reasoning about keys for XML', *Inf. Syst.* **28**(8), 1037–1063.

Buneman, P., Fan, W., Siméon, J. & Weinstein, S. (2001), 'Constraints for semi-structured data and XML', *SIGMOD Record* **30**(1), 47–54.

Buneman, P., Fan, W. & Weinstein, S. (2000), 'Path constraints in semistructured databases', *JCSS* **61**(2), 146–193.

Chamberlin, D., Florescu, D. & Robie, J. (n.d.), 'XQuery Update Facility, W3C Working Draft, July 2006', http://www.w3.org/TR/xqupdate/.

Clark, J. & DeRose, S. (n.d.), 'XML path language (XPath) Version 1.0, W3C Recommendation, Nov. 1999', http://www.w3.org/TR/xpath.

Duta, A. C., Barker, K. & Alhajj, R. (2004), Convrel: relationship conversion to xml nested structures, *in* 'SAC', ACM Press, pp. 698–702.

Fan, W. (2005), XML constraints, *in* 'DEXA Workshops', pp. 805–809.

Fan, W. & Libkin, L. (2002), 'On XML integrity constraints in the presence of DTDs', *J. ACM* **49**(3), 368–406.

Fan, W. & Siméon, J. (2003), 'Integrity constraints for XML', *JCSS* **66**(1), 254–291.

Hara, C. & Davidson, S. (1999), Reasoning about nested functional dependencies, *in* 'PODS', pp. 91–100.

Hartmann, S. (2001), 'On the implication problem for cardinality constraints and functional dependencies', *Ann.Math.Art.Intell.* **33**, 253–307.

Hartmann, S. & Link, S. (2003), More functional dependencies for XML, *in* 'ADBIS', number 2798 *in* 'LNCS', Springer, pp. 355–369.

Hartmann, S. & Link, S. (2007*a*), Numerical keys for XML, *in* 'WoLLIC', number 4576 *in* 'LNCS', Springer, pp. 203–217.

Hartmann, S. & Link, S. (2007*b*), Unlocking keys for XML trees, *in* 'ICDT', number 4353 *in* 'LNCS', Springer, pp. 104–118.

Hartmann, S., Link, S. & Schewe, K.-D. (2006), 'Functional and multivalued dependencies in nested databases generated by record and list constructor', *Ann.Math.Art.Intell.* **46**, 114–164.

Hartmann, S., Link, S. & Trinh, T. (2007), Efficient reasoning about XFDs with pre-image semantics, *in* 'DASFAA', number 4443 *in* 'LNCS', Springer, pp. 1070–1074.

Hartmann, S. & Trinh, T. (2006), Axiomatising functional dependencies for XML with frequencies, *in* 'FoIKS', number 3861 *in* 'LNCS', Springer, pp. 159–178.

Imamura, T., Dillaway, B. & Simon, E. (n.d.), 'XML encryption syntax and processing, W3C Recommendation, Dec. 2002', http://www.w3.org/TR/xmlenc-core/.

Kay, M. (n.d.), 'XSL transformations (XSLT) Version 2.0 W3C Recommendation, Jan. 2007', http://www.w3.org/TR/xslt20/.

Kleiner, C. & Lipeck, U. W. (2001), Automatic generation of xml dtds from conceptual database schemas., *in* 'GI Jahrestagung (1)', pp. 396–405.

Kudrass, T. & Krumbein, T. (2003), Rule-based generation of xml dtds from uml class diagrams, *in* 'ADBIS', pp. 339–354.

Lee, D., Mani, M., Chiu, F. & Chu, W. W. (2002), Net & cot: translating relational schemas to xml schemas using semantic constraints, *in* 'CIKM '02', pp. 282–291.

Lenzerini, M. & Nobili, P. (1990), 'On the satisfiability of dependency constraints in entity-relationship schemata', *Inf. Syst.* **15**(4), 453–461.

Liddle, S., Embley, D. & Woodfield, S. (1993), 'Cardinality constraints in semantic data models', *Data & Knowledge Engineering* **11**, 235–270.

Liu, C. & Li, J. (2006), Designing quality xml schemas from e-r diagrams., *in* 'WAIM', pp. 508–519.

Liu, C., Liu, J. & Guo, M. (2003), On transformation to redundancy free xml schema from relational database schema., *in* 'APWeb', pp. 35–46.

Liu, C., Vincent, M. W. & Liu, J. (2006), 'Constraint preserving transformation from relational schema to xml schema', *World Wide Web* **9**(1), 93–110.

Lv, T. & Yan, P. (2006), Mapping relational schemas to XML DTDs with constraints, *in* 'IMSCCS (2)', pp. 528–533.

Pigozzo, P. & Quintarelli, E. (2005), An algorithm for generating xml schemas from er schemas., *in* 'SEBD', pp. 192–199.

Suciu, D. (2001), 'On database theory and XML', *SIGMOD Record* **30**(3), 39–45.

Thalheim, B. (1991), *Dependencies in Relational Databases*, Teubner.

Thalheim, B. (1992), Fundamentals of cardinality constraints, *in* 'ER', number 645 *in* 'LNCS', Springer, pp. 7–23.

Thompson, H., Beech, D., Maloney, M. & Mendelsohn, N. (n.d.), 'XML Schema Part 1: Structures second edition, W3C Recommendation, Oct. 2004', http://www.w3.org/TR/xmlschema-1/.

Vianu, V. (2003), 'A web odyssey: from Codd to XML', *SIGMOD Record* **32**(2), 68–77.

Vincent, M., Liu, J. & Liu, C. (2004), 'Strong functional dependencies and their application to normal forms in XML', *TODS* **29**(3), 445–462.

Vincent, M. W., Liu, J. & Liu, C. (2003), 'Redundancy free mappings from relations to xml.', pp. 55–67.

Wang, C., Lo, A., Alhajj, R. & Barker, K. (2005), Novel approach for reengineering relational databases into xml., *in* 'ICDE Workshops', p. 1284.