XML Constraints: Specification, Analysis, and Applications

Wenfei Fan
LFCS, School of Informatics, University of Edinburgh
wenfei@inf.ed.ac.uk

Abstract

This paper reviews the recent developments in specification languages, static and run-time analyses as well as applications of integrity constraints for XML.

1 Introduction

XML data [11], just like traditional databases, may be specified by both *type constraints* and *integrity constraints*. XML Schema [31], for example, supports types and integrity constraints (*e.g.,* keys and foreign keys). The distinction between types and integrity constraints is rather arbitrary [14]. A "type" in XML, such as a DTD [11] considered as an extended context-free grammar, typically means a restriction on the element structure of a document, and does not relate data values across elements. The term "integrity constraint" in XML is often used to mean extensions of relational integrity constraints such as keys, foreign keys, and functional dependencies, which depend primarily on the equality of data values. We follow this convention and focus on integrity constraints for XML, subsequently referred to as *XML constraints* in this paper. There has also been a host of work on XML types and typechecking (see, *e.g.,* [3, 25, 32]), which will not be discussed in this paper.

The study of integrity constraints has been recognized as one of the most important yet challenging areas of XML research [32]. For relational databases, constraints are essential to schema design, query optimization, efficient storage and access methods (see, *e.g.,* [1]). For all the reasons that relational constraints are important, XML constraints are also needed. In addition, unlike the relational model, XML is being increasingly used as the prime standard for data exchange and as a uniform model for data integration, and XML data is commonly used by remote applications that know nothing a priori about the semantics of the data. With this come many new applications of XML constraints in, *e.g.,* assisting in preserving the semantics of the data during transformation to a new model, detecting inconsistencies in integrated data, and rewriting queries from XML to SQL and vice versa [2, 8, 9, 10, 16, 18, 19, 23, 26, 33].

Generalizing relational constraints to XML constraints is nontrivial. The hierarchical nature of XML data calls for not only (a) *absolute constraints* that hold on an entire XML document, such as dependencies found in relational databases, but also (b) *relative constraints* that only hold on sub-documents, beyond what we encountered in traditional databases. Thus we need richer languages for specifying XML constraints than we do in the relational setting.

To make effective use of XML constraints it is often necessary to reason about them, both at compile-time (for consistency and implication analyses) and at run-time (for incremental constraint checking). Since XML constraints are more complicated than relational dependencies, the static analyses of XML constraints are more intriguing than their relational counterparts. Add to this the difficulty introduced by XML types (*e.g.,* DTDs), which are far more complex than relational schemas. Thus static analyses that may be trivial for relational databases are intractable or even undecidable for XML [5, 13, 21]. Run-time checking of XML constraints is also harder and it depends on whether the XML data is in a native store [2, 9] or shredded into relations [10].

We next review XML constraint languages (Section 2), static and run-time analyses of XML constraints (Section 3), and applications of XML constraints in practice (Section 4); research directions are also identified (Section 4). This paper is by no means a comprehensive survey: a number of related articles are not referenced due to the space constraint.

2 Constraint Languages for XML

Keys, foreign keys, functional and embedded dependencies have been proposed and studied for XML.

Keys and foreign keys. Although a number of dependency formalisms were developed for relational databases, keys and foreign keys are the ones used most often in practice and are supported by the SQL standard [24]. They provide a mechanism by which one can uniquely identify a tuple in a relation and refer to a tuple from another relation.

Primitive keys and foreign keys are supported by XML DTDs [11] through the use of ID and IDREF attributes, respectively. While an ID attribute can uniquely identify an element within an XML document, it is more like an internal "pointer" rather than a key. (a) ID attributes are not well scoped. In contrast to keys, they are unique within the entire document rather than among a designated set of elements. Thus one cannot, for example, allow a *student* element and

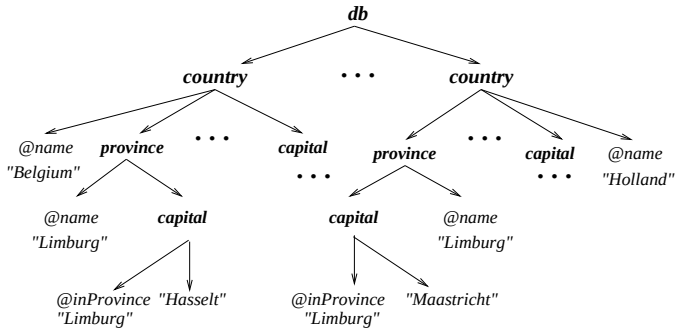

Figure 1. An example XML document

a *person* element to use the same SSN as an ID. (b) Using ID attributes as keys means that we are limiting ourselves to *unary* keys and to using attributes rather than elements. (c) One can specify at most one ID attribute for an element type, while in practice one may want more than one key. Similarly, IDREF attributes are also restricted to be unary and are not well scoped; thus it is not surprising that a *taking* IDREF attribute of a *student* may point to a fellow *student* rather than a *course* element as it intends to.

The hierarchical nature of XML data demands more than mild generalizations of relational constraints. To illustrate this let us consider the XML document depicted in Fig. 1. For each country, the document lists its provinces and capitals of provinces. Suppose we want to define keys for countries and provinces. One can state that country *name* is a key for *country* elements. It is also tempting to say that *name* is a key for *province*, but as shown in Fig. 1, both Holland and Belgium have a province named Limburg. On the other hand, *name* is indeed a key of provinces *relative* to *country* elements: it is extremely unlikely that two provinces of the same country would have the same name. This motivated the notion of *relative keys* proposed in [12]:

$$\varphi \;=\; P_c(P_t\,(S \rightarrow \epsilon)),$$

where P_c and P_t are regular (or XPATH) expressions, and S is a set of regular (or XPATH) expressions; P_c, P_t and S are referred to as the *context* path, the *target* path, and the *key* paths of φ, respectively. An XML document T satisfies φ if and only if for any node v reached from the root of T via P_c, and for any nodes v_1, v_2 reachable from v via P_t, if from v_1 and v_2 the same (set of) values are reached via S, then v_1 and v_2 must be the same node. Here the value equality is defined on a textual encoding of XML elements (*i.e.,* subtrees; see [12] for the details). The relative key φ applies to all subtrees rooted at nodes reached via the context path P_c. In a special case when P_c is the empty path, it applies to the entire document and is referred as an *absolute key*. For example, the following are keys on the XML document of Fig. 1 (in an abused XPATH syntax):

φ_1 = ϵ(//*country*({*name*} $\rightarrow \epsilon$)),
φ_2 = //*country*(//*province*({*name*} $\rightarrow \epsilon$)),
φ_3 = //*country*(//*capital* ({*inProvince*} $\rightarrow \epsilon$)),

where φ_1 is an absolute key in which $P_c = \epsilon$, P_t = //*country*, and S = {*name*}; it says that *name* is a key of *country* elements in the entire document. The rest are *relative keys* specified for subtrees rooted at *country* elements (note that P_c is //*country* in φ_2 and φ_3). They assert that for each country *c*, *name* is a key of all *province* descendants of *c*, and *inProvince* is a key of all *capital* descendants of *c*.

Similarly, a *relative foreign key* is defined as [5]:

$$\phi \;=\; P_c(P_1[L_1] \subseteq P_2[L_2]),$$

where P_1, P_2 are regular (XPATH) expressions, $[L_1]$ and $[L_2]$ are lists of regular (XPATH) expressions, and P_c is the context path as above. An XML document T satisfies ϕ if and only if for any node v reached from the root of T via P_c, for any v_1 reached from v via P_1 there exists v_2 reached from v via P_2, such that the list of values reached via L_1 from v_1 equal those reached via L_2 from v_2. When the context path P_c is ϵ, ϕ is called an *absolute foreign key*. For example, the following is a relative foreign key on Fig. 1:

ϕ_4 = //*country*(//*capital*[*inProvince*] $\subseteq$ *province*[*name*]),

which says that for each *country c*, *inProvince* is a foreign key of all *province* descendants of *c* referring to *name* of *province* elements in the *same sub-document* rooted at *c*.

A key φ (resp. foreign key ϕ) is *unary* if the set S of key paths (resp. L_1, L_2) is a singleton. Let Σ_0 denote $\{\varphi_1, \varphi_2, \varphi_3, \phi_4\}$; all the XML constraints in Σ_0 are unary.

Relative constraints capture information that cannot possibly be specified by absolute constraints.

Constraints in XML Schema [31]. XML Schema adopted [30] a variation of absolute keys and foreign keys proposed in [12, 5]. It defines constraints using a fragment of XPATH, and imposes a uniqueness restriction on the values reached via paths in S in keys (resp. L_1, L_2 in foreign keys).

Functional dependencies. Generalizations of relational functional dependencies (FDs) have also been defined for XML [8, 33]. FDs of [8] are of the form $S_1 \rightarrow S_2$, where S_i is a set of paths for $i \in [1, 2]$. The semantics of these FDs is given in terms of relational FDs over a relational coding of XML data. The coding maps paths in an XML document T to relational tuples, referred to as tree tuples; based on the coding, T satisfies $S_1 \rightarrow S_2$ if and only if for any tree tuples t_1, t_2 in the coding of T, if $t_1.S_1 = t_2.S_1$ and $t_1.S_1$ is well defined, then $t_1.S_2 = t_2.S_2$. XML FDs of [33], referred to XFDs, are interpreted over XML trees directly. Both forms of XML FDs are absolute constraints and are defined in terms of simple paths, *i.e.,* sequences of XML tags. Neither [8] nor [33] considered *relative* XML FDs or FDs defined with more general path expressions that allow the Kleene star (in regular expressions), wildcard or descendants (in XPATH).

XICs. Recall that relational embedded constraints are syntactically defined as (see, *e.g.,* [1], for the details):

$$\forall x_1 \ldots \forall x_n \, (\alpha(x_1, \ldots, x_n) \rightarrow \exists y_1 \ldots \exists y_m \, \beta(x_1, \ldots, x_n, y_1, \ldots, y_m)),$$

where α, β are conjunctions of relational atoms.

A generalization of relational embedded constraints, referred to as XICs, was introduced for XML in [17]:

$$\forall x_1 \ldots \forall x_n \, (B(x_1, \ldots, x_n) \rightarrow \bigvee_{i \in [1,l]} \exists y_{i,1} \ldots \exists y_{i,k_i} \, C_i(x_1, \ldots, x_n, y_{i,1}, \ldots, y_{i,k_i})),$$

where B, C_i are conjunctions of simple XPATH expressions interpreted as binary predicates. Relative functional and inclusion dependencies for XML can be expressed as XICs.

In summary, the language of XICs is the most expressive constraint language studied so far for XML, and it subsumes the language of (relative) keys and foreign keys of [12, 5]. Keys and foreign keys of XML Schema are a variation of absolute constraints of [12, 5]. FDs of [8, 33] generalize a very restricted set of absolute keys of [12], analogous to the extension of relational keys to relational FDs.

3 The Analyses of XML Constraints

We emphasize the static analyses of XML constraints, including the consistency and implication analyses and XML constraint propagation. We also discuss run-time constraint checking, on which little work has been done.

Consistency and Implication Analyses. These are the central problems for reasoning about XML constraints.

An XML specification (schema) typically consists of a DTD D (type) and a set Σ of XML constraints. The *consistency problem* is to determine, given an XML specification (D, Σ), whether or not there exists an XML document that both conforms to the DTD D and satisfies the constraints Σ.

The *implication problem* in the absence of DTDs (resp. in the presence of DTDs) is to determine, given a set Σ of XML constraints and another constraint φ (resp. and a DTD D), whether or not for any XML document T, if T satisfies Σ (resp. and conforms to D) then T satisfies φ.

Keys and foreign keys. In relational databases, the consistency analysis is trivial: one can specify arbitrary (primary) keys and foreign keys in SQL, without worrying about consistency. In the context of XML, however, there is intricate interaction between XML constraints and DTDs, and the interaction complicates the consistency analysis [21].

To illustrate the interaction, recall the set Σ_0 of unary XML keys and foreign keys given earlier on the *country* document of Fig. 1, and consider the DTD D_0 below:

```
<!ELEMENT db       (country+)>
<!ELEMENT country  (province+, capital+)>
<!ELEMENT province (capital, city*)>
```

The DTD imposes *type constraints* on XML documents: each country has a nonempty list of provinces followed by a nonempty list of province capitals, and for each province we specify its capital and perhaps other cities. In addition, each country and province has an attribute *name*, and each capital has an attribute *inProvince* (not shown in D_0).

The specification (D_0, Σ_0), which might look reasonable at first, is actually inconsistent! To see this, let T be an XML tree that satisfies the (D_0, Σ_0). The constraints φ_2, φ_3 and ϕ_4 say that for any subtree rooted at a country c, the number of its *capital* elements is at most the number of *province* elements among c's descendants. The DTD says that each *province* has a *capital* element as a child and that each *country* has at least one *capital* child. Thus, the number of *capital* descendants of c is larger than the number of *province* descendants of c, which contradicts the previous bound.

This tells us that the consistency analysis for XML is more intricate than its relational counterpart. Indeed, the following complexity bounds for XML consistency analysis [5, 21] are self-evident. (a) it is *undecidable* for a class $\mathcal{L}_A$ of *absolute* keys and foreign keys defined in terms of very restricted XPATH expressions; (b) it is already *intractable* for only *unary* keys and foreign keys in $\mathcal{L}_A$; (c) it is *undecidable* even for *unary relative* keys and foreign keys; (d) it is PSPACE-hard for absolute *unary* keys and foreign keys defined in terms of regular path expressions. Upper bounds and several tractable special cases have also been given in [5, 21]. See [7] for a recent survey.

For the implication analysis in the presence of DTDs, it is known [5] that for any class $\mathcal{C}$ of XML constraints that contains unary (primary) keys and foreign keys, if its associated consistency problem is K-hard for some complexity class K that contains DLOGSPACE, then its implication problem is *co*K*-hard*. Thus the lower bound for the implication analysis of $\mathcal{C}$ follows from its consistency counterpart. Several upper bounds have also been given in [5, 21].

For the implication analysis in the absence of DTDs, it is already *undecidable* for *absolute* keys and foreign keys defined in terms of simple paths [22]. It is in *PTIME* and is *finitely axiomatizable* (*i.e.,* there exists a finite set of inference rules sound and complete for implication) for *relative keys* alone defined in terms of a fragment of XPATH [13].

Constraints in XML Schema. A number of lower bounds [5, 21] for the consistency and implication analyses of *absolute* keys and foreign keys carry over to XML-Schema constraints, including the undecidability and intractability results mentioned above. Furthermore, the additional uniqueness condition imposed by XML Schema on the definition of keys makes the analyses even harder. For example, for XML-Schema keys alone, the consistency problem is already intractable [6], while in contrast, it is in linear time for even *relative* keys in the absence of this condition [7].

Functional dependencies. For the implication analysis of FDs of [8], it was shown that the problem is in PTIME or

is coNP-complete under various restricted DTDs [8]. For generic DTDs, the implication problem remains open.

XICs. While a full treatment of the consistency and implication problems for XICs of [17] is not yet in place, Chase and Backchase algorithms were developed using XICs [19], which emphasize the natural connection between the implication analysis and query reformulation, along the same lines as the Chase algorithm for relational dependencies (see, *e.g.,* [1]). A completeness result and several (sufficient) termination conditions were established [19], which assure that the algorithms terminate and find minimal query reformulations for certain restricted XICs, XML queries and views. The connection between the implication analysis of XICs and the containment analysis for a fragment of XPATH in the presence of XICs was explored in [17].

Constraint Propagation. Constraint propagation is another static analysis, which has proven important in designing normalized relational schema for storing XML data [16].

The propagation problem from XML *keys to relational* FDs is to determine, given a mapping σ from XML to relations of a schema R, a set Σ of XML keys and a relational FD ψ on R, whether or not $\Sigma \models_\sigma R : \psi$, *i.e.,* whether for any XML document T, if T satisfies Σ then its relational representation $\sigma(T)$ satisfies ψ. The ability to determine $\Sigma \models_\sigma R : \psi$ is useful in checking the consistency of predefined relational schema for storing XML data.

We also want to *find a minimum cover* of FDs: given a universal relation U, a set Σ of XML keys, and a mapping σ from XML to U, compute a minimum cover F_{mc} for the set F^+ of all FDs on U propagated from Σ via σ. Here F_{mc} is a non-redundant subset of (the exponentially large) F^+ such that all the FDs of F^+ can be derived from F_{mc}. With this one can design a relational storage for XML data as follows: start with a rough (universal) schema, and then decompose U into a BCNF or 3NF guided by F_{mc} [16]. This is analogous to designing relational database schema [1].

For XML keys of [12] and a practical mapping language defined in terms of XPATH, PTIME algorithms and inference systems have been developed [16] for computing XML key propagation and finding a minimum cover. A followup [15] of [16] considered XML FD propagation, for which, however, even the decidability is not yet known.

The topic of constraint propagation deserves a full treatment. (a) For XML data stored in relations, constraint propagation may play an important role in checking XML constraints and in querying and updating XML data *using a RDBMS*. (b) Constraint propagation yields a connection between XML and traditional data models at the semantic level, and is likely to be effective in defining *information-preserving transformations* between XML and databases. However, no work has studied constraint propagation from relations to XML, or between XML and other data models.

Run-Time Analysis. The *constraint checking* problem is to determine, given an XML document T and a set Σ of XML constraints, whether or not T satisfies Σ. A related problem is *incremental constraint checking*, which is to determine, given Σ, T that satisfies Σ, and an XML update Δ on T, whether or not the updated document $\Delta(T)$ satisfies Σ. It is desirable to check constraints incrementally by limiting the checking to the area affected by the updates and thus minimizing unnecessary recomputation.

For XML data in its native format, (incremental) constraint checking has been studied [10, 9, 2]. A constraint checking technique was studied in data integration [10] for (relative) XML keys and inclusion constraints. It treats both DTDs and XML constraints in a uniform framework via constraint compilation, based on a nontrivial extension of attribute grammars. For incremental constraint checking, a technique was developed in [9] for a large class of XML constraints defined in terms of XPATH, based on a code generator that produces incremental checking code. Another incremental technique was proposed in [2] for relative XML keys and foreign keys, based essentially on an extension of incremental techniques for attribute grammars.

4 Applications of XML Constraints

We next review applications in which XML constraints have proven useful, and suggest topics for future research.

Schema design: normal forms for XML. Normal forms for XML schema were first studied in [8], which proposed a generalization of relational BNCF, referred to as XNF. The definition of XNF and the algorithms for lossless decomposition into XNF [8] are based on XML FDs, along the same lines as their relational counterparts. These provide a guideline for producing well-designed XML, and are useful for preventing XML update anomalies, among other things. Another XML normal form was studied in [33], based on XFDs.

As mentioned earlier, constraint propagation from XML to relations has been used in designing semantic-preserving and normalized relational schema for storing XML data [16].

Querying XML data. Answering XML queries using a RDBMS is important both for XML data stored in relations and for XML views exported from relations. Efficient techniques for XML query rewriting and optimization have been developed in [19, 18] by encoding certain XICs and XQuery queries in terms of relational queries and disjunctive embedded dependencies [28], and by using Chase and Backchase algorithms. These have led to a number of theoretical results [19] and a prototype system, MARS [18]. Another optimization technique for XML-SQL query translation was studied in [26] by capitalizing on (relational) constraints.

Updating XML data. Based on the incremental constraint checking technique of [9] mentioned above, a constraint-maintenance tool has been developed and is being used at

Lucent Technologies. Techniques for the consistency analysis also have obvious application in XML editors, *e.g.,* XML Spy [4]. There is, however, much more to be done in this line of research. First, the complexity of (incremental) constraint checking has not been studied. Second, while it is common to find XML data stored in relations, there has not been a full treatment of (incremental) constraint checking for XML data stored in a RDBMS.

Data exchange and integration. One often wants to integrate data *w.r.t.* a predefined XML schema. Given distributed source database schemas $R_1, \ldots, R_n$ and a target XML schema (D, Σ), we want to define a mapping σ such that, given source instances $I_1, \ldots, I_n$, $\sigma(I_1, \ldots, I_n)$ is an XML document that both conforms to the fixed DTD D and satisfies the given XML constraints Σ. In such *schema-directed XML integration*, it is nontrivial to guarantee that D and Σ are both satisfied. To this end the constraint compilation and checking technique of [10] has proven effective.

In data integration applications, one may be able to answer queries even without a complete description of the mapping σ. As pointed out by [27], exploiting integrity constraints may be useful for overcoming the incompleteness of the specification. Although there has been an enormous amount of work in this regard for relational and nested relational data (*e.g.,* [20, 29]), developing XML integration systems for querying incomplete information using constraints is an important topic for future work.

Data cleaning. When overlapping information from multiple sources is integrated, inconsistencies and conflicts in the data may emerge as violations of integrity constraints on the integrated data. XML constraint repair may yield an effective technique for cleaning integrated XML data, as suggested by preliminary work [23], and it may also lead to automated tools for repairing Web sites with broken links.

Acknowledgments. The author would like to thank Peter Buneman, Michael Benedikt and Philip Bohannon for helpful comments. The author is supported in part by EPSRC GR/S63205/01, EPSRC GR/T27433/01 and NSFC 60228006.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Abrão, B. Bouchou, M. Alves, D. Laurent, and M. Musicante. Incremental constraint checking for XML documents. In *XSym*, 2004.

[3] N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with data values: typechecking revisited. *JCSS*, 66(4):688–727, 2003.

[4] Altova. XML Spy. `http://www.altova.com`.

[5] M. Arenas, W. Fan, and L. Libkin. On verifying consistency of XML specifications. In *PODS*, 2002.

[6] M. Arenas, W. Fan, and L. Libkin. What's hard about XML Schema constraints? In *DEXA*, 2002.

[7] M. Arenas, W. Fan, and L. Libkin. Consistency issues in XML databases. In *Inconsistency Tolarance*, pages 15–41. Spinger-Verlag, 2005.

[8] M. Arenas and L. Libkin. A normal form for XML documents. *TODS*, 29:195–232, 2004.

[9] M. Benedikt, G. Brun, J. Gibson, R. Kuss, and A. Ng. Automated update management for XML integrity constraints. In *PLANX*, 2002.

[10] M. Benedikt, C. Y. Chan, W. Fan, J. Freire, and R. Rastogi. Capturing both types and constraints in data integration. In *SIGMOD*, 2003.

[11] T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C, 1998.

[12] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39(5):473–487, 2002.

[13] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. *Information Systems*, 28(8):1037–1063, 2003.

[14] P. Buneman, W. Fan, and S. Weinstein. Interaction between path and type constraints. *TOCL*, 4(4):530–577, Oct. 2003.

[15] Y. Chen, S. Davidson, C. Hara, and Y. Zheng. Constraints preserving schema mapping from XML to relations. In *VLDB*, 2003.

[16] S. Davidson, W. Fan, C. Hara, and J. Qin. Propagating XML constraints to relations. In *ICDE*, 2003.

[17] A. Deutsch and V. Tannen. Containment and integrity constraints for XPath. In *KRDB*, 2001.

[18] A. Deutsch and V. Tannen. MARS: A system for publishing XML from mixed and redundant storage. In *VLDB*, 2003.

[19] A. Deutsch and V. Tannen. Reformulation of XML queries and constraints. In *ICDT*, 2003.

[20] R. Fagin, P. Kolaitis, and L. Popa. Data exchange: getting to the core. In *PODS*, 2003.

[21] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *JACM*, 49(3):368–406, May 2002.

[22] W. Fan and J. Siméon. Integrity constraints for XML. *JCSS*, 66(1):256–293, 2003.

[23] S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Repairs and consistent answers for XML data with functional dependencies. In *Xsym*, 2003.

[24] International Standard ISO/IEC 9075-2:2003(E). Information technology: Database languages, SQL Part 2 (Foundation, 2nd edition), 2003.

[25] N. Klarlund, T. Schwentick, and D. Suciu. XML: Model, schemas, types, logics, and queries. In *Logics for Emerging Applications of Databases*. Springer-Verlag, 2003.

[26] R. Krishnamurthy, R. Kaushik, and J. Naughton. Efficient XML-to-SQL query translation: Where to add the intelligence. In *VLDB*, 2004.

[27] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[28] L. Popa, A. Deutsch, A. Sahuguet, and V. Tannen. A chase too far? In *SIGMOD*, 2000.

[29] L. Popa, Y. Velegrakis, R. Miller, M. Hernández, and R. Fagin. Translating Web data. In *VLDB*, 2002.

[30] H. Thompson. Personal communication, 2002.

[31] H. Thompson et al. XML Schema. W3C Recommendation, Oct. 2004. `http://www.w3.org/TR/xmlschema1`.

[32] V. Vianu. A Web odyssey: From Codd to XML. In *PODS*, 2001.

[33] M. Vincent, J. Liu, and C. Liu. Strong functional dependencies and their application to normal forms in XML. *TODS*, 29:445–462, 2004.