XML Design for Relational Storage

Solmaz Kolahi
University of Toronto
solmaz@cs.toronto.edu

Leonid Libkin
University of Edinburgh
libkin@inf.ed.ac.uk

ABSTRACT

Design principles for XML schemas that eliminate redundancies and avoid update anomalies have been studied recently. Several normal forms, generalizing those for relational databases, have been proposed. All of them, however, are based on the assumption of a native XML storage, while in practice most of XML data is stored in relational databases.

In this paper we study XML design and normalization for relational storage of XML documents. To be able to relate and compare XML and relational designs, we use an information-theoretic framework that measures information content in relations and documents, with higher values corresponding to lower levels of redundancy. We show that most common relational storage schemes preserve the notion of being well-designed (i.e., anomalies- and redundancy-free). Thus, existing XML normal forms guarantee well-designed relational storages as well. We further show that if this perfect option is not achievable, then a slight restriction on XML constraints guarantees a "second-best" relational design, according to possible values of the information-theoretic measure. We finally consider an edge-based relational representation of XML documents, and show that while it has similar information-theoretic properties with other relational representations, it can behave significantly worse in terms of enforcing integrity constraints.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Normal Forms*; H.1.1 [**Models and Principles**]: Systems and Information Theory—*Information Theory, Value of Information*

General Terms

Design, Management, Theory

Keywords

XML data, Design, Relational storage, Functional dependencies, Equality-generating dependencies, Information theory

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

1. INTRODUCTION

Database design and normalization, which originated with early papers by Codd [12, 13, 14], is one of the classical areas of relational database theory, and a standard textbook topic. With a recent shift to XML as a data model, many of classical database subjects have been reexamined in the XML context, among them design and normalization [4, 35, 37, 36, 20, 38]. The goal of normalization is to eliminate redundancies from a database or an XML document, and by doing so, eliminate or reduce potential update anomalies. With that goal in mind, XML normal forms have been introduced [4, 38, 36] and proved to eliminate redundancies in data storage [5].

However, all the work on XML design was making an assumption of a native XML storage for reasoning about redundancies and anomalies. While native XML storage facilities exist [31, 30, 23, 1], many XML processing systems take advantage of relational databases to store and query XML data [21, 39, 32, 34, 18]. The issues of storing and querying relational representations of XML have been studied extensively (refer to [26] for a survey).

So the rationale for good XML designs that can be found in the literature applies to the storage model that is not the prevalent one in practice. Thus, a natural question to ask is: *Do the existing principles of XML design apply when one stores XML in relational databases?* And, in case there is a mismatch between the native XML representation and a relational one, how can one adjust XML design principles to guarantee well-designed relational representation of XML documents?

As we try to formulate these questions in a more precise way, one issue arises immediately: how do we compare XML designs and designs of their relational translations? After all, the notions of redundancies, updates, queries, etc. are rather different in these two worlds. To overcome this problem, we use an *information-theoretic approach* to database designs proposed in [5] and further developed in [25]. The idea of this approach is that it measures, in a way independent of features such as updates and queries, the information content of data, as an entropy of a suitably chosen probability distribution. The higher this information content is, the less redundancy the design carries. This measure applies across different data formats and integrity constraints, and allows us to reason about and compare data designs over different data models.

The values of the information-theoretic measure are real numbers between 0 and 1, with 0 meaning "completely redundant information" and 1 corresponding to no redun-

dancy at all. Good designs are those where all data items have measure 1. It is known, for example, that over relational databases this corresponds to BCNF (Boyce-Codd Normal Form) [2, 24, 28], and over XML documents, to a normal form XNF introduced in [4], if only functional dependencies are involved.

When XML documents are stored as relations, constraints may not remain in the same forms, e.g. XML keys may change to functional dependencies over the relational schema [17]. However, the information-theoretic approach applies to *any* kind of constraints, and as our first result, we are able to show that the normal form XNF corresponds *precisely* to the perfect designs of relational translations of XML. For this result, we impose fairly mild conditions on translations of XML; in fact for most translations used in the literature [33, 21] the result holds.

While one often tries to achieve a perfect design, in practice it is not always possible. For example, if we deal with relational databases and functional dependencies, the perfect design that eliminates all redundancies is BCNF, but if one needs to enforce all the functional dependencies at the same time, a decomposition into BCNF may not exist [2]. In that case, one usually tries to obtain a 3NF (3rd Normal Form) design; in fact 3NF is much more commonly used in practice than BCNF. It was recently shown that 3NF can be explained information-theoretically as the best relational normal form achievable if all constraints are preserved [25]. Using the information-theoretic measure, one can also characterize it as a normal form always guaranteeing values of the measure at least $\frac{1}{2}$, which is the highest value that one can guarantee if the preservation of functional dependencies is important.

Here we show that there is a simple XML design criterion that guarantees this second-best relational design. Namely, our XML design criterion says that every constraint violating XNF should be *relative* [10, 3], i.e. restricted to some element type of a DTD that occurs under the scope of a Kleene star.

Thus, our results suggest the following guidelines for XML design if one stores XML documents in relations:

1. try to achieve the normal form XNF (using, e.g., algorithm from [4]);
2. if that fails, try replacing all XNF-violating constraints by relative ones.

This way one guarantees good design not only of an XML document itself, but also of its relational storage by removing redundancies.

Our final result deals with a relational representation of documents in which we essentially store the document as a tree (the edge relation) [21]. This representation shares some of the information-theoretic characteristics with those more commonly used, but we show that it may require arbitrarily many more relational joins for enforcing XML constraints even for well-designed documents in XNF.

Organization. The paper is organized as follows. In Section 2 we define relational and XML constraints and DTDs. In Section 3 we give a brief overview of normal forms BCNF, 3NF for relational databases, and XNF for XML. In Section 4, we give an overview of the information-theoretic measure, explain how it characterizes good designs. The precise formal definition is in the appendix. In Section 5 we overview relational translations of XML and XML constraints. In Section 6, we show that perfect XML designs (i.e., XNF) correspond precisely to perfect designs of relational representations. In Section 7 we find conditions on XML documents guaranteeing the best non-perfect relational designs, akin to 3NF. In Section 8 we discuss the edge representation, and in Sections 9 and 10 we give conclusions and directions for future work.

2. NOTATIONS

Relational schemas and instances. A relational schema, usually written as S, is a set of relation names. With each m-ary relation $R \in S$, we associate a set of attributes $attr(R)$. We assume that all data values in a database instance are from a countably infinite domain, $\mathbb{N}_+$ (the set of positive natural numbers) for simplicity. Therefore, an instance I of S assigns to each m-attribute relation R in S a finite subset $I(R)$ of $\mathbb{N}_+^m$. The *active domain* of I, denoted by $adom(I)$ is the set of all elements of $\mathbb{N}_+$ that occur in I. The set of *positions* in an instance I of S, denoted by $Pos(I)$, is the set $\{(R, t, A) \mid R \in S,\ t \in I(R) \text{ and } A \in attr(R)\}$.

Schemas may contain *integrity constraints*. We denote such schemas by (S, Σ), where S is a set of relation names and Σ is a set of constraints. In this paper, we deal with constraints such as equality-generating dependencies (EGDs), and functional dependencies (FDs) as special cases of EGDs. An EGD is an expression of the form

$$\forall\,(R_1(\bar{x}_1)\ \wedge\ \ldots\ \wedge\ R_m(\bar{x}_m) \rightarrow x = y)\,,$$

where $\forall$ represents the universal closure of the formula, $x, y \in \bar{x}_1 \cup \ldots \cup \bar{x}_m$, and there is an assignment of variables to columns such that each variable occurs only in one column, and x, y are assigned to the same column. We assume that FDs are of the form $X \rightarrow Y$, where X, Y are nonempty sets of attribute names.

If Σ is a set of constraints, then Σ^+ denotes the set of all constraints implied by it, and $inst(S, \Sigma)$ stands for the set of all instances of S satisfying Σ. We write $inst_k(S, \Sigma)$ for the set of instances $I \in inst(S, \Sigma)$ with $adom(I) \subseteq [1, k]$.

DTDs and XML trees. Assume that we have the following disjoint sets: *El* of element names, *Att* of attribute names, *Str* of possible values of string-valued attributes. All attribute names start with the symbol @.

A DTD D is defined to be $D = (E, A, P, R, r)$, where

- $E \subseteq El$ is a finite set of element types;
- $A \subseteq Att$ is a finite set of attributes;
- P is a set of rules $\tau \rightarrow P_\tau$ for each $\tau \in E$, where P_τ is a regular expression over $E - \{r\}$;
- R assigns a subset of A of attribute names to each element $\tau \in E$;
- $r \in E$ is the root element.

For simplicity, we do not consider `PCDATA` elements in XML trees since they can be represented by attributes.

An *XML tree* is a finite rooted directed tree $T = (N, G)$, where N is the set of nodes, and G is the set of edges, together with

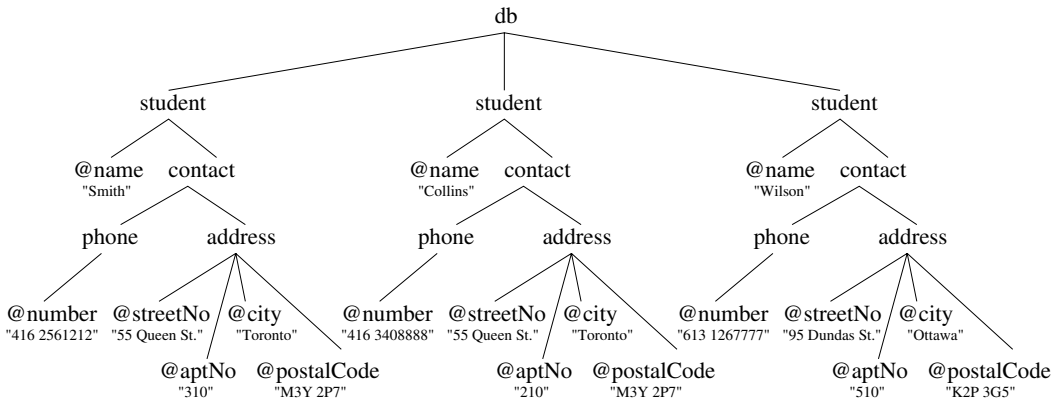

Figure 1: An XML tree.

1. a labeling function $\lambda : N \to El$;
2. an attribute function $\rho_{@a} : N \to Str$ for each $@a \in Att$.

We say tree T *conforms* to DTD $D = (E, A, P, R, r)$, written as $T \models D$, if

- the root of T is labeled r;
- for every $x \in N$ with $\lambda(x) = a$, the word $\lambda(x_1) \ldots \lambda(x_n)$ is in the language defined by P_a where $x_1, \ldots, x_n$ are children of x in order;
- $@l \in R(a)$ iff the function $\rho_{@l}$ is defined on x.

The set of *positions* in an XML document is intuitively the set of places where values (i.e. attribute values) occur. Formally, for a tree $T = (N, G)$ that conforms to DTD D, the set of positions is defined as the set $\{(x, @l) \mid x \in N,\ @l \in R(\lambda(x))\}$, and is denoted by $Pos(T)$.

Given a DTD $D = (E, A, P, R, r)$, an *element path* q is a word in the language E^*, and an *attribute path* is a word of the form $q.@l$, where $q \in E^*$ and $@l \in A$. An element path q is *consistent* with D if there is a tree $T \models D$ that contains a node reachable by q; if the nodes reachable by q have attribute $@l$, then the attribute path $q.@l$ is consistent with D. The set of all paths consistent with a DTD D is denoted by $paths(D)$. The last element type that occurs on a path q is called $last(q)$.

A *functional dependency* over a DTD D [4] is an expression of the form $\{q_1, \ldots, q_n\} \to q$, where $q, q_1, \ldots, q_n \in paths(D)$.

To define satisfaction of functional dependencies, we need a notion of *tree tuples* in XML documents. Given an XML tree $T = (N, G)$ such that $T \models D$, a tree tuple [4] is intuitively a subtree of T rooted at r containing at most one occurrence of every path. Then satisfaction is defined in the usual way: if two tree tuples agree on all the paths $q_1, \ldots, q_n$, then they must agree on q.

The precise definition requires a bit of care since a tree tuple may not be defined on some paths (as tree tuples have at most one occurrence of every path, and may have zero occurrences). Let $\perp$ represents such missing values. A tree tuple is formally defined as a mapping $t : paths(D) \to N \cup Str \cup \{\perp\}$ such that if for an element path q whose last letter is a, we have $t(q) \neq \perp$, then $t(q) \in N$ and $\lambda(t(q)) = a$; if q' is a prefix of q, then $t(q') \neq \perp$ and $t(q')$ lies on the path from the root to $t(q)$ in T; if $@l$ is defined for $t(q)$ and its value is $v \in Str$, then $t(q.@l) = v$.

Then a tree T satisfies an FD $\{q_1, \ldots, q_n\} \to q$ if for any two tree tuples t_1, t_2 in T, whenever $t_1(q_i) = t_2(q_i) \neq \perp$ for all $i \in [1, n]$, then $t_1(q) = t_2(q)$.

If in an FD $\{q_1, \ldots, q_n\} \to q$, for some i the path q_i is an element path, and for all $j \in [1, n], j \neq i$, the path q_i is a prefix of q_j, then we say that such an FD is *relative* (more precisely, relative to q_i).

Example 1. Consider the XML tree of Figure 1 that conforms to DTD $D = (E, A, P, R, db)$, where

$$
\begin{aligned}
&E = \{db, student, contact, address, phone\}\\
&A = \{@name, @streetNo, @aptNo, @city, @postalCode,\\
&\quad @number\}\\
&P = \{db \to student^*,\ student \to contact,\\
&\quad contact \to address^*,\ address \to \epsilon,\ phone \to \epsilon\}\\
&R(student) = \{@name\}\\
&R(address) = \{@streetNo, @aptNo, @city, @postalCode\}\\
&R(phone) = \{@number\}\\
&R(db) = R(contact) = \emptyset
\end{aligned}
$$

This XML tree satisfies the following constraints: (1) for each *student*, no more than one *address* is kept with a single *postalCode*, and (2) the value of *postalCode* uniquely determines *streetNo* and *city*. These constraints can be formulated as the following FDs:

$$db.student,\ db.student.contact.address.@postalCode \to db.student.contact.address \quad (1)$$

$$db.student.contact.address.@postalCode \to db.student.contact.address.@city \quad (2)$$

The first FD is an example of a relative FD, where the constraint holds within each fixed element *student*.

3. OVERVIEW OF NORMALIZATION

A normal form specifies a set of syntactic conditions over the constraints defined for a database that will lead to less redundant instances. We briefly review some normal forms

defined for relational and XML data and refer the reader to surveys [7, 22, 9], texts [2, 24, 28], and papers [4, 38] for additional information.

Relational normal forms. A schema (S, Σ) is in BCNF if for every relation name R in it and every nontrivial FD $X \to Y$ over attributes of R, X is a key of R. Prime attributes are those that belong to a candidate (minimal) key. A schema (S, Σ) is in 3NF if for every relation name R in it and every nontrivial FD $X \to Y$ over attributes of R, either X is a key, or every attribute in $Y - X$ is prime.

Given a single-relation schema (R, Σ) and some normal form $\mathcal{NF}$, a lossless $\mathcal{NF}$-decomposition is a set of schemas (R_j, Σ_j), $j \in J$, such that each (R_j, Σ_j) is in $\mathcal{NF}$, and for every $I \in inst(R, \Sigma)$ we have $\pi_{attr(R_j)}(I) \models \Sigma_j$ and furthermore $I = \bowtie \{\pi_{attr(R_j)}(I) \mid j \in J\}$. Such a decomposition is called *dependency-preserving* if $\left(\bigcup_j \Sigma_j\right)^+ = \Sigma^+$. It is well-known that both 3NF and BCNF admit lossless decompositions, which in the case of 3NF can be guaranteed to be dependency-preserving. In the case of BCNF dependency preservation is not always possible (consider a schema with attributes A, B, C and FDs $AB \to C$ and $C \to A$).

A normal form for XML. Given a DTD D and a set Σ of FDs over D, $(D, \Sigma)^+$ is the set of all FDs implied by (D, Σ). An FD is called *trivial* if it belongs to $(D, \emptyset)^+$. We say that (D, Σ) is in *XML Normal Form (XNF)* [4] if for every nontrivial FD $X \to q.@l$ in $(D, \Sigma)^+$, the FD $X \to q$ is also in $(D, \Sigma)^+$.

Intuitively, this condition prevents redundancies among values of attributes $@l$ of q. It was shown in [4] that if relational databases are translated into XML documents (say, as instances of DTDs $r \to t^*$, where t has all the attributes of a relation R), then XNF coincides with BCNF. Thus, it is a natural extension of BCNF to XML documents. For further results on XNF, and its justification, see [4, 5].

We can see an example of XNF violation by FD (2) in Example 1. While the path on the left-hand side *db.student.contact.address.@postalCode* implies the attribute path *db.student.contact.address.@city*, it does not imply the element path *db.student.contact.address*.

4. INFORMATION THEORY AND NORMALIZATION

Since we shall be comparing XML designs and relational designs, we would like to work in a framework that applies across several data models and is independent of concepts such as query languages and updates (which are not yet as fully understood for XML as for relational world), and concepts such as dependency preservation (which has not been adequately explored in the XML context).

Such framework is provided by the classical information theory and its central concept of entropy which measures information content. Information theory has recently been used to characterize well-known relational normal forms, such as BCNF, 4NF, and 3NF, as well as XML normal forms [5, 25]. Given a database schema S, a set of integrity constraints Σ and an instance I of (S, Σ), the information-theoretic measure, introduced in [5], assigns a number to every position p in the instance that contains a data value. This number, which is called *relative information content* with respect to constraints Σ and is written as $\text{RIC}_I(p|\Sigma)$, ranges between 0 and 1 and shows how much redundancy is carried by position p. Numbers close to 0 mean high redundancy, while numbers close to 1 mean no or low redundancy for the data value in position p.

EXAMPLE 2. *(see [5]) Consider relation $R(A, B, C)$ with the set of FDs $\Sigma = \{A \to B\}$, and three instances I_1, I_2, and I_3 of (R, Σ) shown in Figures 2(a), 2(b), and 2(c) respectively. Let p_1, p_2, and p_3 denote the position of the gray cell in the instances. We can observe that the information content of the gray cell decreases as it becomes more redundant by adding tuples to the instances.*

To define this measure intuitively, suppose that we have n positions (data elements) in a relational or XML instance I, whose values are drawn from a domain of size k (which we may assume without loss of generality to be the interval $[1, k]$). We want to measure, on average, how much information is contained in position p with respect to all other sets X of positions — or, in other words, how much we can derive about p from values in positions in X. When we can derive the value in p unambiguously, the information content will be 0; when we cannot say anything about it, it will be 1.

This in turn is measured as follows. Suppose X is a subset of $Pos(I) - \{p\}$. Suppose the values of positions in X are lost, and then somehow restored from the set $[1, k]$. We measure how much information this gives us about the value of p. This is done in a standard information-theoretic fashion, by computing an entropy of a certain distribution[1]. Then $\text{RIC}_I^k(p|\Sigma)$ is the average of such entropy over all such sets X.

We want the information content to be a value in $[0, 1]$. It is known that the maximum entropy for a discrete distribution on k elements is $\log k$ [15]. So we consider the ratio $\text{RIC}_I^k(p|\Sigma)/\log k$, and then define the relative information content of position p as

$$\text{RIC}_I(p|\Sigma) = \lim_{k \to \infty} \frac{\text{RIC}_I^k(p \mid \Sigma)}{\log k}.$$

A key result of [5] is that this limit exists for all reasonable classes of constraints (e.g., all those definable in first-order logic, such as functional, multi-valued, join, etc. dependencies).

The following definition is from [5] and it applies across different data formats, including relational and XML data.

DEFINITION 1. *A database schema (S, Σ), with a set of constraints Σ, is* well-designed *if for every instance I of (S, Σ) and every position p in I, we have $\text{RIC}_I(p|\Sigma) = 1$.*

In other words, a schema is well-designed if in every instance, no position has any redundancy at all.

Here we concentrate on designs guided by functional dependencies. For them, well-designed schemas have been precisely characterized in [5].

FACT 1. *(see [5])*

1. *If S is a relational schema and Σ is a set of functional dependencies, then (S, Σ) is well-designed if and only if it is in BCNF.*

[1]The precise definition of this distribution and the definition of entropy are in the appendix.

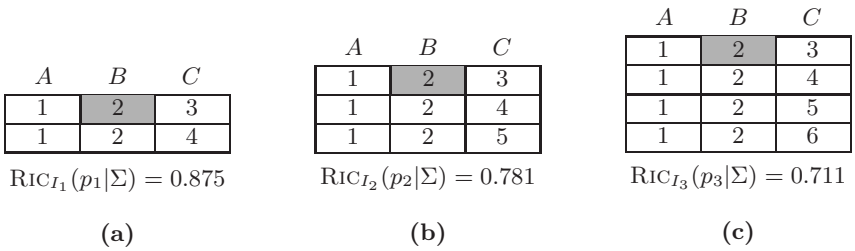

Figure 2: Information content vs redundancy.

2. *If D is a DTD and Σ is a set of XML functional dependencies, then (D, Σ) is well-designed if and only if it is in XNF.*

We also note that the framework applies well beyond functional dependencies. For example, [5] used it to characterize designs based on multi-valued and join dependencies, and we shall use it soon for schemas for equality-generating dependencies (EGDs).

Some popular relational normal forms, such as 3NF, are not well-designed according to our definition, because they allow some amount of redundancy to compensate for preserving all the constraints. A way to measure and compare information contents over multiple schemes satisfying a particular condition, such as a normal form, was introduced in [25].

Let $\mathcal{C}$ be some condition on relational schemas, e.g., BCNF or 3NF. Now consider all instances I of these schemas (S, Σ), and all the possible values $\text{RIC}_I(p|\Sigma)$.

DEFINITION 2. *The* guaranteed information content *of a condition $\mathcal{C}$ is the largest number $c \in [0, 1]$ such that $\text{RIC}_I(p|\Sigma) \geq c$ for all such instances I and positions p. It is denoted by* $\text{GIC}(\mathcal{C})$. *Formally,*

$$\text{GIC}(\mathcal{C}) \;=\; \inf\{\text{RIC}_I(p|\Sigma)\},$$

where I ranges over instances of schemas (S, Σ) that satisfy $\mathcal{C}$, and p ranges over positions in I.

Using this, one can characterize 3NF. In fact, it was noticed long ago that 3NF designs can differ significantly [40], and in general those accepted as good ones are the ones produced by the standard 3NF synthesis algorithm by Bernstein [8]. In fact such a subset of 3NF, which we denote by 3NF^+, can be characterized information-theoretically as follows:

FACT 2. *(see [25])* $\text{GIC}(3\text{NF}^+) = \frac{1}{2}$. *Moreover, if $\mathcal{C}$ is any other normal form that guarantees dependency-preserving decompositions, then* $\text{GIC}(\mathcal{C}) \leq \frac{1}{2}$.

Note that we can reformulate Fact 1 as $\text{GIC}(\text{BCNF}) = 1$. This means that if we cannot achieve the maximum information content equal to 1 for all positions, the next best thing that we guarantee is the value of the information-theoretic measure equal to $\frac{1}{2}$.

5. RELATIONAL TRANSLATION OF XML

The main goal of this paper is to show how a good XML design can result in having a less redundant relational storage for XML documents. Redundancies occur when DTDs permit adding more redundant values, or redundant tree tuples, to XML documents. In particular, they occur when element types occur under the scope of a Kleene star in the DTD. This is indeed what is required for the worst cases of redundancy, so for lower bounds we can safely concentrate on DTDs that basically model nested relations, i.e. non-recursive disjunction-free DTDs, where each element type appears once or under a Kleene star in the production rule of its parent.

We shall use the *inlining* technique [33] as our XML-to-relational mapping scheme. While the inlining technique is not the only [21, 39, 32, 6, 19, 18] or necessarily the best mapping scheme (see [26] for a survey), it produces the most natural relational schema for DTDs that are essentially nested relations. But our results are more general: they apply to any other mapping scheme that produces a similar relational schema for nested-relation-like DTDs.

Given a DTD, the basic idea of the inlining mapping is that separate relations are created for the root and the element types that appear under a star, and the other element types are inlined in the relations corresponding to their parents. Each relation corresponding to an element type has an ID attribute that is a key for that relation as well as a parent ID attribute that is a foreign key pointing to the parent of that element in the document. All the attributes of a given element type in the DTD become attributes in the relation corresponding to that element type.

For example, the relational schema for storing XML documents conforming to the DTD in Example 1 would be

$$\begin{array}{l} student(\underline{stID}, name, \underline{conID}) \\ address(\underline{addID}, conID, postalCode, streetNo, aptNo, city) \\ phone(\underline{phID}, conID, number) \end{array}$$

Key attributes are underlined, and the following foreign keys also hold: $address[conID] \subseteq_{FK} student[conID]$ and $phone[conID] \subseteq_{FK} student[conID]$.

The inlining schema generation can be formally captured as follows.

DEFINITION 3. *Given a DTD $D = (E, A, P, R, r)$, we define the* inlining *of D to be a relational schema S, where*

- *$S = \{R_e \mid e \in E$, and e^* occurs in $P_{e'}$ for some $e' \in E$ or e occurs in $P_r\}$, and*
- *there is a mapping $\sigma : E \to S$ recursively defined as*
$$\sigma(e) = \begin{cases} R_e & \text{if } R_e \in S \\ \sigma(e'), \text{ where } e \text{ occurs in } P_{e'} & \text{if } R_e \notin S \end{cases}$$
such that for each $R_e \in S$,

$$attr(R_e) = \begin{array}{l} \bigcup_{e' \in \sigma^{-1}(R_e)} (\{e'ID\} \cup A(e')) \\ \cup \{e'ID \mid e \text{ occurs in } P_{e'}\} \end{array}.$$

Then given an XML tree $T = (N, G)$ conforming to D and satisfying Σ, we can straightforwardly *shred* it into an instance I_T of relational schema S, the inlining of D. Note that we can use node identifiers in N for values of ID attributes when populating I_T. The precise definition of this transformation is omitted here.

It is easy to observe that the FDs defined over a DTD do not necessarily translate into FDs over the relational schema, simply because the paths involved in an XML functional dependency may not all occur in a single relation. Therefore, we need to join different relations to enforce the integrity constraints that are now in the form of *equality-generating dependencies* (EGDs).

For example, the FDs in Example 1 would translate into the following EGDs. Note that EGD (4) is an FD, but EGD (3) is not.

$$\begin{array}{ll} \forall & student(s,n,c) \ \wedge \ address(a,c,pc,st,apt,ct) \ \wedge \\ & student(s,n,c) \ \wedge \ address(a',c,pc,st',apt',ct') \\ & \rightarrow \ a = a', \end{array} \quad (3)$$

$$\begin{array}{ll} \forall & address(a,c,pc,st,apt,ct) \ \wedge \\ & address(a',c',pc,st',apt',ct') \ \rightarrow \ ct = ct'. \end{array} \quad (4)$$

We can show that every constraint expressed in the form of an FD for XML can be written as an EGD over the inlining relational storage. More formally,

PROPOSITION 1. *For every DTD D and set of XML functional dependencies Σ defined over D, there is a set of EGDs Σ_E over the relations of S, the inlining of D, such that for every XML tree T conforming to D, the tree T satisfies Σ if and only if I_T satisfies Σ_E.*

Note that there are also some key and foreign key constraints $\Sigma_{K,FK}$ over the ID attributes of S, as shown in previous example. We then call (S, Σ_S) the *inlining translation* of (D, Σ), where $\Sigma_S = \Sigma_E \cup \Sigma_{K,FK}$.

6. RELATIONAL TRANSLATIONS PRESERVE PERFECT DESIGNS

We know that if we want a perfectly non-redundant design for XML, we should try to achieve XNF [4]. XML is most commonly stored in relational databases, in order to take advantage of fast relational query engines and well-developed storage facilities. We therefore need to study the design and normalization not only for XML per se, but also for the relational storage of XML documents. Here we study the effect of XNF normalization on the relational storage, and in the next section we will see how the relational storage looks, in terms of redundancy, for non-perfect XML designs.

We observe that the inlining mapping preserves a good XML design by showing that the information content of data values will remain maximum after transforming XML data into relational storage. Note that the FDs over XML data will become EGDs over the relational storage.

Let D be a DTD, Σ be a set of XML functional dependencies defined over D, and (S, Σ_S) be the inlining translation of (D, Σ). Consider an XML tree T conforming to D and satisfying Σ. Let I_T be the instance of (S, Σ_S) that is obtained by shredding T into relations of S. Now every position p in T is naturally mapped into a unique position $\delta(p)$ in I_T.

DEFINITION 4. *We say that an inlining translation (S, Σ_S) of (D, Σ) is* well-designed *iff for every XML tree T conforming to D and satisfying Σ and every position p in T, we have* $\text{RIC}_{I_T}(\delta(p)|\Sigma_S) = 1$.

In other words, in positions corresponding to positions from the XML document, there is no redundancy whatsoever in the shredded documents according to the translation of XML constraints.

THEOREM 1. *The following are equivalent for an XML specification (D, Σ) and its inlining translation (S, Σ_S):*

1. *(D, Σ) is well-designed (or, equivalently, is in XNF);*
2. *(S, Σ_S) is well-designed.*

This means that in order to ensure a non-redundant relational storage for our XML data, we need to have an XNF design. In other words, XNF achieves nonredundant design no matter what type of storage – native or relational – is used.

7. RELATIVE CONSTRAINTS AND GOOD DESIGNS

Like in the relational case, bad XML designs may lead to very low information contents for positions in the relational storage. In fact, the information content of a position in a relational storage of an XML document can potentially be arbitrarily low.

PROPOSITION 2. *For every $\varepsilon > 0$, we can find an XML design (D, Σ) with inlining translation (S, Σ_S) and an XML tree T conforming to D and satisfying Σ with a position p, such that for the corresponding position $\delta(p)$ in I_T, we have* $\text{RIC}_{I_T}(\delta(p)|\Sigma_S) < \varepsilon$.

To avoid the possibility of having such a high redundancy, we need some restrictions that guarantee a reasonable information content for all positions in the relational storage of XML. We showed in the previous section that an XNF design corresponds to maximum information content for the relational storage, but is it always possible to achieve XNF?

Recall that in the relational context it is not always possible to achieve a dependency-preserving perfect design. Therefore to guarantee dependency preservation, we may have to tolerate some redundancy, at most equal to one half of the maximum information content, and this is exactly what a good 3NF (i.e., $3NF^+$) normalization gives us.

Here we want to show that if a dependency-preserving XNF design is not achievable, then a simple restriction on FDs defined for XML guarantees the second best information content for the relational storage of an XML document. The restriction is simply that all FDs should satisfy XNF or be *relative* to an element that occurs under the scope of a Kleene star. Then the information content of all positions of the relational storage of the XML document will be at least $\frac{1}{2}$.

Let $D = (E, A, P, R, r)$ be a DTD, Σ be a set of XML functional dependencies defined over D, and (S, Σ_S) be the inlining translation of (D, Σ).

DEFINITION 5. *We say that an FD* $\{q_1, \ldots, q_n\} \to q \in \Sigma$ *is relative under the Kleene star if*

- q_i *is an element path for some* $i \in [1, n]$,
- *for all* $j \in [1, n]$, q_i *is a prefix of* q_j, *and*
- *for some* p *which is a prefix of* q_i *and* $\tau \in E$, $last(p)$ *occurs under a Kleene star in* P_τ.

An example of FD that violates this condition is FD (2) in Example 1. We refer to such FDs as *absolute* or *global* FDs.

We now extend the definition of guaranteed information content for relational storage of XML documents. Let $\mathcal{C}$ be some condition on XML functional dependencies defined over a DTD, e.g. XNF or relative under the Kleene star. Now consider all XML trees T conforming to some DTD D and satisfying FDs Σ, such that FDs in Σ are of type $\mathcal{C}$. The *guaranteed information content* of a condition $\mathcal{C}$ for inlining translation, written as $\text{GIC}_{inl}(\mathcal{C})$ is the largest number $c \in [0, 1]$ such that for all such trees T and position p in T,

$$\text{RIC}_{I_T}(\delta(p)|\Sigma_S) \geq c,$$

where I_T is the shredding of T into the inlining translation (S, Σ_S) of (D, Σ), and $\delta(p)$ is the position in I_T to which p is mapped. Formally,

$$\text{GIC}_{inl}(\mathcal{C}) \;=\; \inf\{\text{RIC}_{I_T}(\delta(p)|\Sigma_S)\},$$

where T ranges over trees conforming to D and satisfying FDs Σ of type $\mathcal{C}$, and p ranges over positions in T.

Using this, we can reformulate Theorem 1 as $\text{GIC}_{inl}(XNF) = 1$. Now let *relative* denote the condition of being relative under star. Then we can formally state the main result of this section:

THEOREM 2. $\text{GIC}_{inl}(relative) \geq \frac{1}{2}$.

In other words, if we manage to design an XML document in such a way that there is no global FDs, then the redundancy of each data value in the relational storage of the XML document would not be worse than $\frac{1}{2}$.

We can explain this result more intuitively by looking at update anomalies that could happen in the relational storage of an XML document. Most database management systems disallow updates that violate key constraints, but the only mechanism to enforce FDs or EGDs would be through writing assertions. In the absence of global or absolute FDs on the XML side, the possibility of FD or EGD violation due to a bad update in the relational storage will be restricted to a small portion of the entire database. Informally speaking, if we are to numerically evaluate the possibility of having update anomalies, our results state that by having global FDs, the relational storage of an XML document could be exponentially more prone to anomalies, compared to the case when we are restricted to XNF and relative FDs.

8. A DIFFERENT MAPPING SCHEME

Beside inlining, there are other XML-to-relational data and query mapping schemes [21, 39, 32, 6, 19, 18], among them being the *Edge* representation [21], which is used as a basis for many XML query translation techniques. Here we would like to study this mapping scheme from two points of view: 1) redundancy and information content, and 2) the complexity of enforcing integrity constraints.

In the Edge representation, an XML tree is viewed as an edge-labeled graph. Each element-to-element and element-to-attribute edge of the tree has a tuple in *Edge* table, and for each data value in the tree, there is a tuple in *Value* table associating the node identifier to a value. The relational storage for any XML tree, regardless of its schema, has the following relations:

$$Edge(\underline{source, target}, label),$$
$$Value(\underline{vid}, val).$$

In the original definition of this schema [21], there are two more attributes in the *Edge* relation: *ordinal*, which specifies the ordinal of the *target* among children of the *source*, and *flag* which specifies whether the tuple corresponds to an element-to-element edge or an element-to-attribute edge. We simplify the schema by removing these attributes, as they do not have any effect on the redundancy of data values.

Assume that we have a set *El* of element names, *Attr* of attribute names, and *Str* of all possible string-valued attributes. Then given an XML tree $T = (N, G)$, with labeling function $\lambda : N \to El$ and attribute functions $\rho_{@a} : N \to Str$ for each $@a \in Attr$, we can populate *Edge* and *Value* such that

- for each edge $(x, y) \in G$, there is a tuple $(x, y, \lambda(y)) \in Edge$, and
- for each $x \in N$ and $@a \in Attr$ such that $\rho_{@a}$ is defined for x, there is a tuple $(x, y, @a) \in Edge$ as well as a tuple $(y, \rho_{@a}(x)) \in Value$, where $y \notin N$ is a fresh node identifier.

Other variants of this approach also exist. One of them is the *binary* approach [21, 32], where the *Edge* relation is horizontally partitioned based on attribute *label*. The schema would then have the following relations:

$$B_{label}(\underline{source, target})$$
$$Value(\underline{vid}, val).$$

In this representation, edge labels or element types are not stored as attributes. We can therefore assume that the domain of each attribute is an infinite set, and thus the measure of information content can be directly applied. Here we focus on the binary representation and show that redundancy-wise it looks the same as the inlining representation.

Given a set of FDs Σ defined over a DTD D, the translation of Σ over the binary representation of D will again be a set of EGDs. We denote the *binary translation* of (D, Σ) by (S_B, Σ_B), where Σ_B contains some key and foreign key constraints, as well as some EGDs obtained from Σ. Every T conforming to D and satisfying Σ can be trivially shredded into an instance I_B of (S_B, Σ_B), and each position p in T is mapped to a unique position $\delta_B(p)$ in I_B. Then the definitions of being well-designed and guaranteed information content for binary translation, GIC_{bin}, would be very similar to those of inlining translation. Not surprisingly, the binary translation also preserves a perfect XML design:

THEOREM 3. *The following are equivalent for an XML specification* (D, Σ) *and its binary translation* (S_B, Σ_B)*:*

1. (D, Σ) *is well-designed (or, equivalently, is in XNF);*

2. (S_B, Σ_B) *is well-designed.*

Similarly, to guarantee an information content of $\frac{1}{2}$ for positions in the binary representation of an XML document, it is enough to make sure that all the XML functional dependencies are relative:

THEOREM 4. $\text{GIC}_{bin}(\textit{relative}) \geq \frac{1}{2}$.

This might make us think that, from the design point of view, binary and inlining representations are equivalent. However, we will next show that they differ significantly when it comes to the complexity of enforcing integrity constraints. Here by complexity, we mean the number of joins needed to write a SQL assertion that enforces an EGD translated from an XML functional dependency. Given an XML functional dependency f over a DTD D, we use the notation

$$\#\bowtie^f_{inl} \quad \text{and} \quad \#\bowtie^f_{bin}$$

to denote the number of joins required to enforce the translation of f on the inlining and binary relational representation of D respectively.

Consider for instance FD (2) of Example 1. The translation of this FD over the binary representation of the DTD is an EGD that involves five joins, whereas over the inlining translation, it can be written as an FD requiring only one join as shown by (4).

In fact, we can show that even for key or XNF dependencies, the number of joins needed for the binary representation is never smaller than under inlining, and can be arbitrarily higher.

PROPOSITION 3.

1. *For every DTD D and XML functional dependency f, we have $\#\bowtie^f_{bin} \geq \#\bowtie^f_{inl}$.*

2. *For every $m > 0$, we can find a DTD D and an XML functional dependency f over D, such that*

$$\frac{\#\bowtie^f_{bin}}{\#\bowtie^f_{inl}} > m.$$

9. CONCLUSIONS

We have studied design and normalization for XML documents stored in relations, rather than in native XML storage facilities (which was the assumption of the previous work on XML normalization). To be able to compare relational and XML designs, we applied the information-theoretic framework of [5] that can be used for many different data models, and is independent of data model features such as update and query languages (which are still in the development stage for XML).

The main conclusions are as follows:

1. The XML normal form XNF, proposed in [4] as a generalization of BCNF for XML documents, achieves the best possible design from the point of view of eliminating redundancies in both native and relational storage of XML.

   Note that algorithms for converting XML designs into XNF exist.

2. If XNF is not achievable, the next best possible design is achieved by relativizing all constraints that violate XNF. By relativizing we mean restricting them to the scope of an element that occurs in a DTD with a Kleene star (or under the scope of another element with a Kleene star). When we say "the next best possible design", our criterion is the information content of redundant values in the relational storage of XML documents.

3. The information-theoretic framework allows us to compare different shredding techniques from the point of view of redundancies in XML documents. We show that two popular techniques – inlining [33] and the edge representation [21] – behave in the same way information-theoretically, while the latter can behave arbitrarily worse in terms of enforcing integrity constraints.

10. FUTURE WORK

While the situation with the best possible design from the point of view of eliminating update anomalies has now been completely clarified for both native and relational storage, it is not yet entirely clear how to handle non-perfect designs that do not eliminate all redundancies. Such designs are necessary, and in fact essential, in the relational world, where they guarantee database consistency by means of dependency preservation. This in fact explains why 3NF designs in practice are more popular than BCNF designs. In the XML world, we do not yet have an adequate understanding of the notion of dependency preservation and its impact on XML design, whether for native or relational storage.

Another open issue is whether one can relativize constraints, as suggested in this paper for achieving the best non-XNF design, and do it while preserving XML constraints.

We also would like to extend the idea of using the information-theoretic framework for reasoning about and comparing different shredding techniques for XML documents.

Acknowledgments. We thank Wenfei Fan and Reneé Miller for comments and suggestions. Both authors are supported by a grant from NSERC. Part of this work was done while Kolahi was visiting the University of Edinburgh. Libkin is on leave from the University of Toronto, supported by the European Commission Marie Curie Excellence grant MEXC-CT-2005-024502 and EPSRC grant E005039.

11. REFERENCES

[1] Xyleme XML server & business document management. `http://www.xyleme.com/page/xml_storage/`.

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

[3] M. Arenas, W. Fan, and L. Libkin. On verifying consistency of XML specifications. In *PODS*, pages 259–270, 2002.

[4] M. Arenas and L. Libkin. A normal form for XML documents. *ACM Trans. Database Syst.*, 29:195–232, 2004.

[5] M. Arenas and L. Libkin. An information-theoretic approach to normal forms for relational and XML data. *J. ACM*, 52(2):246–283, 2005.
[6] D. Barbosa, J. Freire, and A. O. Mendelzon. Designing information-preserving mapping schemes for XML. In *VLDB*, pages 109–120, 2005.
[7] C. Beeri, P. A. Bernstein, and N. Goodman. A sophisticate's introduction to database normalization theory. In *VLDB*, pages 113–124, 1978.
[8] P. A. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Trans. Database Syst.*, 1(4):277–298, 1976.
[9] J. Biskup. Achievements of relational database schema design theory revisited. In *Semantics in Databases*, pages 29–54, 1995.
[10] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, and W. C. Tan. Keys for XML. In *World Wide Web*, pages 201–210, 2001.
[11] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *VLDB*, pages 71–81, 1987.
[12] E. F. Codd. Normalized database structure: A brief tutorial. In *ACM SIGFIDET Workshop on Data Description, Access and Control*, 1971.
[13] E. F. Codd. Further normalization of data base relational model. In *Courant Computer Science Symposium 6: Data Base Systems*, pages 33–64, 1972.
[14] E. F. Codd. Recent investigations in relational database systems. In *Information Processing*, pages 1017–1021, 1974.
[15] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & sons, 1991.
[16] M. M. Dalkilic and E. L. Roberston. Information dependencies. In *PODS*, pages 245–253, 2000.
[17] S. Davidson, W. Fan, C. Hara, and J. Qin. Propagating XML constraints to relations. In *ICDE*, pages 543–554, 2003.
[18] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *SIGMOD*, pages 431–442, 1999.
[19] A. Deutsch and V. Tannen. MARS: A system for publishing XML from mixed and redundant storage. In *VLDB*, pages 201–212, 2003.
[20] D. W. Embley and W. Y. Mok. Developing XML documents with guaranteed "good" properties. In *ER'01*, pages 426–441.
[21] D. Florescu and D. Kossmann. Storing and querying XML data using an RDBMS. *IEEE Data Eng. Bulletin*, 22(3):27–34, 1999.
[22] P. C. Kanellakis. Elements of relational database theory. pages 1073–1156, 1990.
[23] C.-C. Kanne and G. Moerkotte. A linear-time algorithm for optimal tree sibling partitioning and approximation algorithms in Natix. In *VLDB*, 2006.
[24] M. Kifer, A. Bernstein, and P. M. Lewis. *Database systems : an application-oriented approach*. Addison-Wesley, 2006.
[25] S. Kolahi and L. Libkin. On redundancy vs dependency preservation in normalization: An information-theoretic study of 3NF. In *PODS*, pages 114–123, 2006.
[26] R. Krishnamurthy, R. Kaushik, and J. F. Naughton. XML-to-SQL query translation literature: The state of the art and open problems. In *XSym'03, LNCS 2824*, pages 1–18, 2003.
[27] T. T. Lee. An information-theoretic analysis of relational databases - part i: Data dependencies and information metric. *IEEE Trans. Software Eng.*, 13(10):1049–1061, 1987.
[28] M. Levene, M. Levene, and G. Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer-Verlag, London, UK, 1999.
[29] M. Levene and G. Loizou. Why is the snowflake schema a good data warehouse design? *Inf. Syst.*, 28(3):225–240, 2003.
[30] Z. H. Liu, M. Krishnaprasad, and V. Arora. Native XQuery processing in oracle XMLDB. In *SIGMOD*, pages 828–833, 2005.
[31] M. Nicola and B. van der Linden. Native XML support in DB2 universal database. In *VLDB*, pages 1164–1174, 2005.
[32] A. Schmidt, M. L. Kersten, M. Windhouwer, and F. Waas. Efficient relational storage and retrieval of XML documents. In *Selected papers from the Third International Workshop WebDB 2000 on The World Wide Web and Databases*, pages 137–150. Springer-Verlag, 2001.
[33] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *VLDB*, pages 302–314, 1999.
[34] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In *SIGMOD*, pages 204–215, 2002.
[35] M. W. Vincent and J. Liu. Functional dependencies for XML. In *APWEB*, pages 22–34, 2003.
[36] M. W. Vincent, J. Liu, and C. Liu. A redundancy free 4NF for XML. In *XSym*, pages 254–266, 2003.
[37] M. W. Vincent, J. Liu, and C. Liu. Strong functional dependencies and their application to normal forms in XML. *ACM TODS*, 29(3):445–462, 2004.
[38] J. Wang and R. W. Topor. Removing XML data redundancies using functional and equality-generating dependencies. In *Australian Database Conference*, pages 65–74, 2005.
[39] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Trans. Inter. Tech.*, 1(1):110–141, 2001.
[40] C. Zaniolo. A new normal form for the design of relational database schemata. *ACM Transactions on Database Systems*, 7(3):489–499, 1982.

APPENDIX

A. DEFINITION OF THE INFORMATION-THEORETIC MEASURE

A.1 Basics of information theory

The main concept of information theory is that of entropy, which measures the amount of information provided by a

certain event. Assume that an event can have n different outcomes s_1, ..., s_n. Then for a probability space $\mathcal{A} = (\{s_1, \ldots, s_n\}, P_\mathcal{A})$, where $P_\mathcal{A}$ is a probability distribution, its entropy is defined as

$$H(\mathcal{A}) \;=\; \sum_{i=1}^{n} P_\mathcal{A}(s_i) \log \frac{1}{P_\mathcal{A}(s_i)}.$$

For probabilities that are zero, we adopt the convention that $0 \log \frac{1}{0} = 0$, since $\lim_{x \to 0} x \log \frac{1}{x} = 0$. It is known that $0 \leq H(\mathcal{A}) \leq \log n$, with $H(\mathcal{A}) = \log n$ only for the uniform distribution $P_\mathcal{A}(s_i) = 1/n$ [15].

We shall also need the concept of conditional entropy of $\mathcal{B}$ assuming $\mathcal{A}$.

Suppose we have two probability spaces

$$\begin{array}{rcl} \mathcal{A} & = & (\{s_1, \ldots, s_n\}, P_\mathcal{A}), \\ \mathcal{B} & = & (\{s'_1, \ldots, s'_m\}, P_\mathcal{B}) \end{array}$$

and probabilities $P(s'_j, s_i)$ of all the events (s'_j, s_i). Note that $P_\mathcal{A}$ and $P_\mathcal{B}$ need not be independent. Then the *conditional entropy of $\mathcal{B}$ given $\mathcal{A}$*, denoted by $H(\mathcal{B} \mid \mathcal{A})$, gives the average amount of information provided by $\mathcal{B}$ if $\mathcal{A}$ is known [15]. If

$$P(s'_j \mid s_i) = \frac{P(s'_j, s_i)}{P_\mathcal{A}(s_i)}$$

are conditional probabilities, then the conditional entropy is defined as

$$H(\mathcal{B} \mid \mathcal{A}) = \sum_{i=1}^{n} \left(P_\mathcal{A}(s_i) \sum_{j=1}^{m} P(s'_j \mid s_i) \log \frac{1}{P(s'_j \mid s_i)} \right).$$

A.2 Relative information content

We now give a detailed definition of relative information content from [5] that was used to justify relational and XML normal forms [5, 25].

Unlike other proposed information-theoretic measures [27, 11, 16, 29] that work only at the level of data, this measure takes into account both data and schema constraints.

Fix a schema S and a set Σ of constraints, and let $I \in \mathit{inst}(S, \Sigma)$. We want to define $\textsc{Ric}_I(p \mid \Sigma)$, the relative information content of a position $p \in \mathit{Pos}(I)$ with respect to the set of constraints Σ.

Formally, we assume that I has n positions (which we enumerate as $1, \ldots, n$), and fix an n-element set of variables $\{v_i \mid 1 \leq i \leq n\}$. Let $\Omega(I, p)$ be the set of all 2^{n-1} vectors $(a_1, \ldots, a_{p-1}, a_{p+1}, \ldots, a_n)$ such that for every $i \in [1, n] - \{p\}$, a_i is either v_i or the value in the i-th position of I. We make this into a probability space $\mathcal{A}(I, p) = (\Omega(I, p), P_u)$ with the uniform distribution $P_u(\bar{a}) = 2^{1-n}$.

We next define conditional probabilities $P_k(a \mid \bar{a})$ that show how likely a is to occur in position p, if values are removed from I according to the tuple $\bar{a} \in \Omega(I, p)$ Let $I_{(a,\bar{a})}$ be obtained from I by putting a in position p, and a_i in position $i \neq p$. A substitution is a map $\sigma : \bar{a} \to [1, k]$ that assigns a value to each a_i which is a variable, and leaves other a_is intact. We let $\mathrm{SAT}^k_\Sigma(I_{(a,\bar{a})})$ be the set of all substitutions σ such that $\sigma(I_{(a,\bar{a})}) \models \Sigma$ and $|\sigma(I_{(a,\bar{a})})| = |I|$ (the latter ensures that no two tuples collapse as the result of applying σ). Then $P_k(a \mid \bar{a})$ is defined as:

$$P_k(a \mid \bar{a}) \;=\; \frac{|\mathrm{SAT}^k_\Sigma(I_{(a,\bar{a})})|}{\sum_{b \in [1,k]} |\mathrm{SAT}^k_\Sigma(I_{(b,\bar{a})})|}.$$

With this, we define $\textsc{Ric}^k_I(p \mid \Sigma)$ as

$$\sum_{\bar{a} \in \Omega(I,p)} \left(\frac{1}{2^{n-1}} \sum_{a \in [1,k]} P_k(a \mid \bar{a}) \log \frac{1}{P_k(a \mid \bar{a})} \right).$$

Since $\sum_{a \in [1,k]} P_k(a \mid \bar{a}) \log \frac{1}{P_k(a \mid \bar{a})}$ measures the amount of information in p, given constraints Σ and some missing values in I, represented by the variables in $\bar{a}$, our measure $\textsc{Ric}^k_I(p \mid \Sigma)$ is the average such amount over all $\bar{a} \in \Omega(I, p)$.

To see that $\textsc{Ric}^k_I(p \mid \Sigma)$ is a conditional entropy, define

$$P'_k(a) \;=\; \frac{1}{2^{n-1}} \sum_{\bar{a} \in \Omega(I,p)} P_k(a \mid \bar{a}) \;.$$

It is a probability distribution on $[1, k]$ (intuitively, it says how likely an element from $[1, k]$ is to satisfy Σ when put in position p, given all possible interactions between p and sets of positions in I). If $\mathcal{B}^k_\Sigma(I, p)$ is the probability space $([1, k], P'_k)$, then $\textsc{Ric}^k_I(p \mid \Sigma)$ is the conditional entropy:

$$\textsc{Ric}^k_I(p \mid \Sigma) \;=\; H(\mathcal{B}^k_\Sigma(I, p) \mid \mathcal{A}(I, p)).$$

Since the domain of $\mathcal{B}^k_\Sigma(I, p)$ is $[1, k]$, we have $0 \leq \textsc{Ric}^k_I(p \mid \Sigma) \leq \log k$. To normalize this, we consider the ratio $\textsc{Ric}^k_I(p \mid \Sigma)/\log k$. The key observation of [5] is that for most reasonable constraints Σ (certainly for all definable in first-order logic), this sequence converges as $k \to \infty$, and we thus define

$$\textsc{Ric}_I(p|\Sigma) \;=\; \lim_{k \to \infty} \frac{\textsc{Ric}^k_I(p \mid \Sigma)}{\log k}.$$