Developing XML Documents with Guaranteed "Good" Properties

D.W. Embley
Department of Computer Science
Brigham Young University
Provo, Utah 84602
USA
embley@cs.byu.edu

W.Y. Mok
Department of Business
Information Systems and Education
Utah State University
Logan, Utah 84322
USA
wmok@cc.usu.edu

Abstract

Many XML documents are being produced, but there are no agreed-upon standards formally defining what it means for complying XML documents to have "good" properties. In this paper we present a formal definition for a proposed canonical normal form for XML documents called *XNF*. XNF guarantees that complying XML documents have maximally compact connectivity while simultaneously guaranteeing that the data in complying XML documents cannot be redundant. Further, we present a conceptual-model-based methodology that automatically generates XNF-compliant DTDs and prove that the algorithms, which are part of the methodology, produce DTDs to ensure that all complying XML documents satisfy the properties of XNF.

1 Introduction

Many DTDs (Document Type Definitions) for XML documents are being produced (e.g. see [XML]), and soon many XML-Schema specifications [XML00] for XML documents will be produced. With the emergence of these documents, we should be asking the question, "What constitutes a good DTD?"[1] We argue that a "good" DTD should guarantee that all complying XML documents are in an agreed-upon form that has two desirable properties: (1) the DTD should have as few hierarchical trees as possible rooted just below the top-level node, and (2) at the same time, the DTD should not allow any of these trees to have redundant data values in XML documents that comply with the DTD. Intuitively, this should ensure that complying XML documents are

[1]Since we do not address issues beyond hierarchical structure in this document, we discuss the issues in terms of DTDs rather than XML-Schemas. Further, since proposed specifications require XML-Schema to include full DTD expressibility, we do so without loss of generality.

compactly connected in as few hierarchies as possible while simultaneously ensuring that no data value in any complying document can be removed without loss of information.

In this paper we formalize these ideas by defining XNF, a normal form for XML documents,[2] and we present a way to generate DTDs that are guaranteed to be in XNF. We assume that the DTDs produced are for XML documents representing some aspect of the real world—those for which conceptual modeling makes sense.[3] Under this assumption, we argue that to produce a "good" DTD for an application A, we should first produce a conceptual-model instance for A and then apply a transformation guaranteed to produce an XNF-compliant DTD.

Although we can guarantee XNF compliance, we cannot guarantee uniqueness. In general, several "good" DTDs, correspond to any given conceptual-model instance. Selecting the best depends on usage requirements and viewpoints that are "in the eye of the beholder." Sometimes these usage requirements or viewpoints should even cause the principles of XNF to be violated, but most of the time XNF-compliance should be compatible with usage requirements and viewpoints. Heuristic "rules of thumb" can go a long way toward resolving this problem of nonuniqueness and can often produce results that are highly satisfactory. We believe, however, that the ultimate resolution should be synergistic. Given heuristic rules and the principles of XNF, a system should work with a user to derive a suitable application DTD. The system can automatically derive reasonable XNF-compliant DTDs. The user may adjust, reject, or redo any of the generated suggestions. The system can check the revisions and report any violations of XNF so that the user is aware of the consequences of the revisions. Iterating until closure is reached, the user can further revise the DTD, and the system can evaluate and provide feedback.

We are aware of only one other research effort that closely parallels our work—namely [BGH00].[4] [BGH00] makes the same argument we make, namely (1) that graphical conceptual-modeling languages offer one of the best—if not the best—human-oriented way of describing an application, (2) that a model instance should be transformed automatically into an XML DTD (or XML schema), and (3) that the transformation should maximize connectivity among the XML components and should minimize the redundancy in complying XML documents. The authors of [BGH00] use Object Role Modeling [Hal99] as their conceptual model and give a set of twelve heuristics for generating the "best" XML schema. What is missing is the guarantee that their transformation achieves maximum connectivity and minimum redundancy. In this paper we use a more generic conceptual model and a simpler set of heuristics to achieve similar results, but the main contribution is the guarantee that complying XML documents satisfy the formal properties XNF.

[2]The idea of XNF is based on nested normal form as defined in [MEN96] and is also related to other similar nested normal forms such as those surveyed in [Mok].

[3]The class of documents we are considering is certainly large, but also certainly not all-inclusive. We do not, for example, consider text documents where the order of textual elements is important.

[4]Others, such as [BR00, CSF00], discuss a UML-to-XML transformations, but they do not investigate properties of conceptual-model generated XML documents. Similarly, OMG's XMI effort [XMI] also provides a way to represent UML in XML, but the effort is devoid of XNF-like guarantees for XML documents.

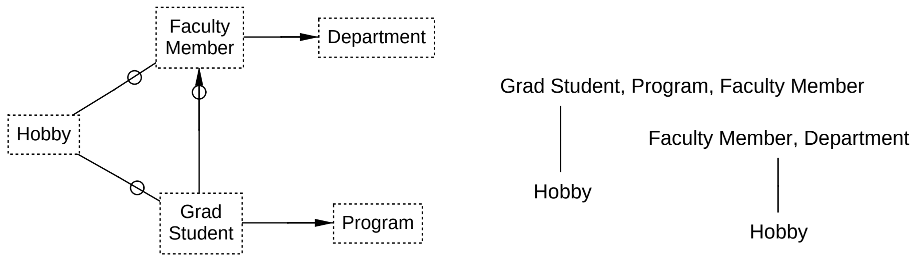
We present our contribution as follows. Section 2 provides motivating examples and foundation definitions. Besides arguing that we can produce DTDs that yield only XNF-compliant XML documents, we also provide examples to show that even for simple applications, it is easy to produce (and thus nontrivial to avoid) DTDs that have redundancy and have more hierarchical clusters than necessary. In Section 3 we present straightforward algorithms that guarantee the properties of XNF for a large number of practical cases. We then show in Section 4, however, that these straightforward algorithms depend on the given conceptual-model instance being in a particular canonical form. Although many practical model instances are naturally specified in the required canonical form, some are not. We therefore explain how to achieve this canonical form Section 4. We then prove that for a given conceptual-model instance in the canonical form, the algorithms in Section 3 yield XNF-compliant DTDs. In Sections 5 and 6 we generalize our approach and give algorithms for producing XNF-compliant DTDs for a more-inclusive set of conceptual-model instances. We conclude in Section 7 and present the status of our implementation.

2 Motivating Example

As a motivating example, consider the conceptual-model diagram in Figure 1(a). Although based on [EKW92], the conceptual modeling approach we present here is generic. Users model the real world by constraint-augmented hypergraphs, which we call *CM hypergraphs* (conceptual-model hypergraphs). Vertices of CM hypergraphs are object sets denoted graphically as named rectangles. The object set *Hobby* in Figure 1(a), for example, may denote the set {*Chess*, *Hiking*, *Sailing*}. In the general conceptual model, edges have two or more connections to object sets, but we restrict ourselves until Section 5 to edges with just two connections. Edges representing functional relationships have arrowheads on the range side. In Figure 1(a), a graduate student enrolls in only one program (e.g. *PhD*, *MS*) and has only one faculty-member advisor. A connection between an object set and a relationship set may be optional or mandatory, denoted respectively by an “O” on the edge near the connection or the absence of an “O.” A faculty member need not have hobbies and need not have advisees, but must be in a department. The inclusion constraints, which for Figure 1(a) are only simple referential integrity constraints, must all hold. For optional participation, the inclusion constraint allows a proper subset, but for mandatory participation, the subset can never be proper. Later in the paper, a triangle with its apex connected to a generalization object set and its base connected to one or more specialization object sets will denote an explicit inclusion constraint—the objects in any specialization object set must be a subset of the objects in the generalization object set.

An XML document has a hierarchical structure with a single root.[5] The set of structures immediately below the single root constitutes a forest of hierarchical trees. It is this forest of trees

[5]We do not concern ourselves with XML documents that have no DTD or XML documents rooted at a DTD subelement.


Faculty Member
Department
Hobby
Grad Student
Program
Grad Student, Program, Faculty Member
Hobby
Faculty Member, Department
Hobby

(a) Sample Diagram (b) Sample Scheme Trees

Figure 1: Simple Faculty-Student-Hobbies Diagram and Scheme Trees

we wish to derive from a conceptual-model instance.

We abstractly represent each tree in this forest as a *scheme tree.*

Definition 1 Let U be a set of object sets. A *scheme tree* is recursively defined as follows:

1. If X is a nonempty multiset[6] of object sets in U, then a tree T with only the root node that contains the object sets in X is a *scheme tree* over X.

2. If X, X_1, ..., X_n are nonempty multisets of object sets in U, and T_1, ..., T_n are scheme trees over X_1, ..., X_n respectively, then a tree T with the root node that contains the object sets in X and the root nodes of the T_i's, $1 \leq i \leq n$, as the children of the root node of T is a *scheme tree* over the union of the multisets X, X_1, ..., X_n. $\Box$

Definition 2 A *textual representation* of a scheme tree T is recursively defined as follows:

1. If T has only the root node, then $(L)*$ is a *textual representation* of T where L is a list of the object sets contained in the root node of T.

2. If T has more than one node, then let T_1, ..., T_n, $n \geq 1$, be the n subtrees of T such that each of the root nodes of the T_i's, $1 \leq i \leq n$, is a child of the root node of T. If L_i is a textual representation of T_i, $1 \leq i \leq n$, and L is a list of the object sets contained in the root node of T, then $(L, L_1, \ldots, L_n)*$ is a *textual representation* of T. $\Box$

Example 1 The two scheme trees in Figure 1(b) cover the model instance in Figure 1(a). Textually written, the scheme trees in Figure 1(b) are *(Grad Student, Program, Faculty Member, (Hobby)*)** and *(Faculty Member, Department, (Hobby)*)**. $\Box$

¿From a forest of scheme trees, we can derive a DTD for any model instance as follows.[7] (1) Select a name N for the root and generate <! *DOCTYPE N* [<*scheme trees*><*data elements*>

[6]Multisets allow us to handle recursion and other multiple references to the same object set.

[7]There are several ways we can represent a scheme tree as a DTD (especially if we use XMI [XMI] or features such as ATTLIST, ID, and IDREF). These, however, can be obtained by transforming the DTD itself. Thus, except to

```
<!DOCTYPE University[
    <!ELEMENT University (
        (Grad_Student, Program, Faculty_Member,
            (Hobby)* )*,
        (Faculty_Member_2, Department,
            (Hobby_2)* )* )>
    <!ELEMENT Grad_Student (#PCDATA)>
    <!ELEMENT Program (#PCDATA)>
    <!ELEMENT Faculty_Member (#PCDATA)>
    <!ELEMENT Hobby (#PCDATA)>
    <!ELEMENT Faculty_Member_2 (#PCDATA)>
    <!ELEMENT Department (#PCDATA)>
    <!ELEMENT Hobby_2 (#PCDATA)>
]>
```

(a) Sample DTD Specification

```
<University>
    <Grad_Student>Pat</Grad_Student>
        <Program>PhD</Program>
        <Faculty_Member>Kelly</Faculty_Member>
        <Hobby>Hiking</Hobby>
        <Hobby>Skiing</Hobby>
    <Grad_Student>Tracy</Grad_Student>
        <Program>MS</Program>
        <Faculty_Member>Kelly</Faculty_Member>
        <Hobby>Hiking</Hobby>
        <Hobby>Sailing</Hobby>
    <Grad_Student>Chris</Grad_Student>
        <Program>MS</Program>
        <Faculty_Member>Kelly</Faculty_Member>
    <Faculty_Member_2>Kelly</Faculty_Member_2>
        <Department>CS</Department>
        <Hobby_2>Chess</Hobby_2>
        <Hobby_2>Skiing</Hobby_2>
    <Faculty_Member_2>Noel</Faculty_Member_2>
        <Department>Math</Department>
        <Hobby_2>Sailing</Hobby_2>
</University>
```

(b) Sample XML Document

Figure 2: Sample DTD and Complying XML Document

] >. (2) To each object-set name that appears more than once in the scheme-tree forest, append a "_2" to the second, a "_3" to the third, etc.[8] Further, to make names comply with DTD requirements, replace any white space in an object-set name by an underscore character. (3) Replace $<$*scheme trees*$>$ by $<!\ ELEMENT\ N\ (<the\ generated\ scheme\!-\!tree\ forest>) >$ where N is the selected name for the root and "*the generated scheme-tree forest*" is a comma-separated list of textual representations of the scheme trees in the scheme tree forest. (4) Replace $<$*data elements*$>$ by a sequence of $<!\ ELEMENT\ N\ (\#PCDATA) >$, one for each object-set name N including object-set names with appended numbers. Occasionally there will be no data for a required element; in this case, we can let the data be an empty string.

Example 2 Figure 2(a) shows a generated DTD for the two scheme trees in Figure 1(b), and Figure 2(b) shows a sample complying XML document. □

provide the one translation we give here, we do not concern ourselves further with this issue, but rather concentrate on generating a forest of XNF-compliant scheme trees.

[8]If necessary, we can specify alternate ways to make object-set names unique. Moreover, we would normally like to have users specify alternate names, which should be mnemonically chosen to indicate the role played by the object set in the context of the application.

Example 3 As motivational examples, consider the following scheme-tree forests.

1. *(Department, (Faculty Member, (Hobby)*, (Grad Student, Program, (Hobby)*)*)*)**
2. *(Faculty Member, Department, (Hobby)*, (Grad Student, Program, (Hobby)*)*)**
3. *(Hobby, (Faculty Member)*, (Grad Student)*)*, (Grad Student, Program, Faculty Member)*, (Department, (Faculty Member)*)**
4. *(Hobby, (Faculty Member, Department)*, (Grad Student, Program, Faculty Member)*)**
5. *(Faculty Member, Department, (Hobby, (Grad Student)*)*)*, (Grad Student, Faculty Member, Program)** □

We claim (and will shortly give the formal basis for showing) that the first two sample scheme-tree forests in Example 3, which each consist of a single tree, can never have a redundant data value in any complying XML document. The third scheme-tree forest, as well as the scheme-tree forest in Figure 1(b), also never allow redundancy, but both have more than one scheme tree and thus neither is as compact as the first two sample scheme trees. The fourth sample scheme-tree forest allows redundancy—since faculty members and grad students are listed repeatedly for each additional *Hobby* in which they participate, department values for faculty members and program values as well as faculty advisors for grad students can be redundant. The first tree of the fifth scheme-tree forest also allows redundancy—whenever faculty members have the same hobbies, all the graduate students that also share these hobbies are listed.

Definition 3 Let M be a CM hypergraph whose object sets and relationship sets are populated with data values such that the populated CM hypergraph is a valid interpretation (i.e., is a model in terms of model theory [AD93]). Let U be the set of object sets in M. Let T be a scheme tree over some subset of U. A *scheme-tree instance over* T is recursively defined as follows:

1. If T has only the root node and the root node contains the nonempty multiset $O = \{O_1, \ldots, O_n\}$ of object sets, then r is a *scheme-tree instance over* T if r is a (possibly empty) set of functions $\{t_1, \ldots, t_m\}$ where each function t_i, $1 \le i \le m$, maps O_j in O to a data value in O_j in M, $1 \le j \le n$.

2. If T has more than one node, then let $T_1, \ldots, T_k$, $k \ge 1$, be the k subtrees of T such that each of the root nodes of the T_i's, $1 \le i \le k$, is a child of the root node of T. Assume the root node of T contains the nonempty multiset $O = \{O_1, \ldots, O_n\}$ of object sets, then r is a *scheme-tree instance over* T if r is a (possibly empty) set of functions $\{t_1, \ldots, t_p\}$ where each function t_i, $1 \le i \le p$, maps O_j in O to a data value in O_j in M, $1 \le j \le n$, and maps T_q to a sheme-tree instance over T_q, $1 \le q \le k$. □

Example 4 The tuples for the first populated scheme tree in Figure 2(b) written as functions, are

{(Grad Student, Pat), (Program, PhD), (Faculty Member, Kelly), {(Hobby, Hiking), (Hobby, Skiing)}}
{(Grad Student, Tracy), (Program, MS), (Faculty Member, Kelly), {(Hobby, Hiking), (Hobby, Sailing)}}
{(Grad Student, Chris), (Program, MS), (Faculty Member, Kelly), {}} □

Definition 4 Let T be a scheme tree for a CM hypergraph M, and let t be a scheme-tree instance over T. A data value v in t is *redundant with respect to a constraint* C that holds for M if when v is replaced in t by some symbol, say $\perp$, where $\perp$ is not in t, C implies $\perp = v$. □

Although many constraints are possible, we consider only functional constraints, multivalued constraints, and inclusion constraints. We further restrict this set to those that are conceptual-model compliant in the sense that they occur naturally within a conceptual-model instance as defined below. This set of constraints is a common standard set that is sufficient for many, if not most, practical cases.

Let T be a scheme tree. We denote the multiset of object sets in T by $Aset(T)$. Let N be a node in T. Notationally, $Ancestor(N)$ denotes multiset that is the the union of object sets in all ancestors of N, including N. A *path* of T is a sequence of nodes from the root node of T to a leaf node of T. Thus, if T has n, $n \geq 1$, leaf nodes, T has n paths. A scheme tree T is *properly constructed* for a CM hypergraph M if every path of T embeds a sequence of some connected edges in M.

Definition 5 Let T be a properly constructed scheme tree for a CM hypergraph M, and let t be a scheme-tree instance over T. Let $X \rightarrow Y$ be a functional edge in M that is contained in a path of T, and let s be a subtuple over XY in t. Let A be an attribute of Y,[9] and let a be the A value in s. Then t *has redundancy with respect to the functional constraint* $X \rightarrow Y$ if the a-value in s appears more than once in t. If such a scheme-tree instance t can exist, we say that T *has potential redundancy with respect to the functional constraint* $X \rightarrow Y$. □

Example 5 Scheme-tree four in Example 3 has potential redundancy with respect to the functional edge *Faculty Member* $\rightarrow$ *Department* because faculty members appear once for each hobby and may participate in several hobbies. Similarly, since grad students also appear once for each hobby and may participate in several hobbies, there is potential redundancy with respect to both the functional constraints *Grad Student* $\rightarrow$ *Program* and *Grad Student* $\rightarrow$ *Faculty Member*.

Definition 6 Let T be a properly constructed scheme tree for a CM hypergraph M, and let t be a scheme-tree instance over T. Let X—Y be an edge in M that is contained in a path of T, and let s be a subtuple over XY in t. Let y be the Y subtuple in s.[10] Then t *has redundancy with respect to the multivalued constraint* X—Y if the y-subtuple in s appears more than once in t. If such a scheme-tree instance t can exist, we say that T *has potential redundancy with respect to the multivalued constraint* X—Y. □

[9] A is Y for the binary case, but in general, Y is a set of attributes.
[10] For the binary case y is a value, but in general, y is a subtuple.

Example 6 The first scheme tree in the fifth scheme-tree forest in Example 3 has potential redundancy with respect to the edge *Hobby* — *Grad Student.* Whenever faculty members have the same hobby, hobby values appear more than once.

Observe that if a scheme tree T has potential redundancy with respect to a functional constraint, then T clearly has redundancy with respect to a multivalued constraint.

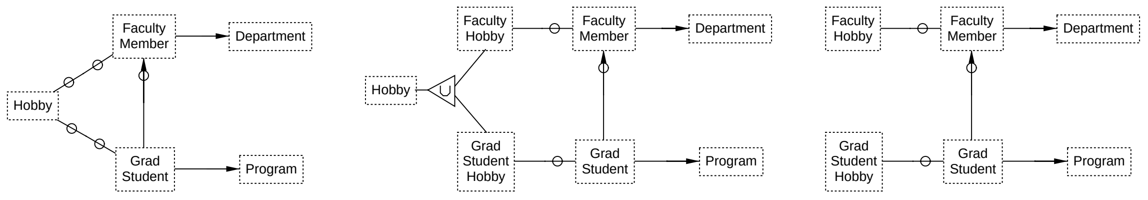
Definition 7 A scheme-tree forest F *corresponds* to a conceptual-model instance M if each tree of F is a properly constructed scheme tree and the union of the edges in all the scheme trees of F covers the edges in M. □

Definition 8 Let F be a scheme-tree forest corresponding to a conceptual-model instance M. Let T be a scheme tree in F with only a root node and only one object set G in the root node. If there exist object sets S_1, ..., S_n within the nodes of F such that the S_i's $(1 \leq i \leq n)$ are specializations of G in M and $G = \cup_{i=1}^{n} S_i$ for all scheme-tree-forest instances for F, the values in G are redundant and T *has potential redundancy with respect to the inclusion constraint* specifying that the S_i's are union specializations of G. □

Example 7 To illustrate inclusion constraints and redundancy with respect to inclusion constraints, we present the CM hypergraphs in Figure 3. Figure 3(a) is the same as Figure 1(a) except that $Hobby$ is optional for both $Faculty$ and $Grad\ Student$. The diagram in Figure 3(a) allows all hobbies for both faculty and students to be listed, not just those shared jointly by at least one faculty member and grad student as is the case for the hobbies in the diagram in Figure 1(a). Using the application model instance in Figure 3(a), however, we cannot include all of the data values in scheme-tree instances for either the first or the second scheme tree in Example 3. This is because the model instance in Figure 3(a) allows hobbies to be listed that are neither faculty hobbies nor student hobbies; thus an "extra" scheme tree is necessary to accommodate these hobbies. If this is not what we want (it probably isn't), we should model our microworld as in Figure 3(b) where the union constraint[11] on the generalization/specialization ensures that $Hobby$ contains only hobbies that are faculty hobbies (contained in $Faculty\ Hobby$) or student hobbies (contained in $Grad\ Student\ Hobby$). Since $Hobby = Faculty\ Hobby \cup Grad\ Student\ Hobby$, $Hobby$ is redundant and we should eliminate it as Figure 3(c) shows. For Figure 3(c), the first and second scheme tree in Example 3 apply, except that the $Hobby$ element associated with $Faculty$ should be $Faculty\ Hobby$ and the $Hobby$ element associated with $Grad\ Student$ should be $Grad\ Student\ Hobby$. With this solution, we eliminate redundancy with respect to inclusion constraints, and we also have better names for XML tags. □

Definition 9 Let M be a CM hypergraph. Let F be a scheme-tree forest corresponding to M. F is in XNF_C if each scheme tree in F has no potential redundancy with respect to a specified

[11]A union symbol inside a generalization/specialization triangle denotes that the generalization object set is a union of the specialization object sets.



(a) All Hobbies (b) *Hobby*—Redundant (c) No ID Redundancy

Figure 3: Illustration for Inclusion Dependencies and ID Redundancy

set of constraints C and F has as few, or fewer, scheme trees as any other scheme-tree forest corresponding to M in which each scheme tree has no potential redundancy with respect to C. When all constraints apply or when the specified set of constraints is clear from the context, we write XNF_C simply as XNF. □

Example 8 We claim (and will later show) that only the first two scheme-tree forests in Example 3 are in XNF. □

3 Binary Algorithm

In Figure 4 We present our first algorithm to generate scheme-tree forests. This algorithm requires the input to be a conceptual-model instance with only binary edges connecting vertices and no explicit generalization/specialization.

Example 9 Consider the model instance in Figure 1(a) as the input to Algorithm 1. If we select *Department* as the root node, we make *Faculty Member* a child of *Department* and designate it as a continuation attribute and then make *Hobby* a child of *Faculty Member*; further since *Faculty Member* is a continuation attribute, we make *Grad Student* another child of *Faculty Member* and designate it as a continuation attribute and then add *Program* in the node with *Grad Student* and finally make *Hobby* a child node of the node containing *Grad Student.* The result is the first scheme tree in Example 3. If we select *Faculty Member* as the starting node, we can generate the second scheme tree in Example 3. If we select *Grad Student* as the starting node, proceed as far as we can, and then select *Faculty Member* from the remaining unmarked nodes and proceed, we can generate the scheme-tree forest in Figure 1(b). Similarly, we can generate the third scheme-tree forest in Example 3, by starting with *Hobby*, then *Grad Student*, and then *Department.* We cannot, however, generate either the fourth or the fifth scheme-tree forest in Example 3. □

Observe from our discussion of Examples 3, 5, 6, and 9 that Algorithm 1 disallows the sample scheme-tree forests that have potential redundancy. Indeed, we claim, and will prove in Section 4

```
*Input*: a binary CM hypergraph $H$
    (with no explicit generalization/specialization).
*Output*: a scheme-tree forest.
Until all vertices and edges have been marked
    If one or more unmarked vertices remain
        Let $V$ be a selected unmarked vertex in $H$;
        Mark $V$ in $H$;
    Else
        Select an unmarked edge $E$;
        Let $V$ be one of the two vertices of $E$;
    Make $V$ the root node of a new scheme tree $T$;
    Designate $V$ in $T$ as a continuation attribute;
    While there is an unmarked edge $E = (A, B)$ in $H$
            such that $A$ is a continuation attribute in $T$
        Mark $E$ in $H$;
        If the $B$-$E$ connection is mandatory, Mark $B$ in $H$;
        If $A \leftrightarrow B$
            Add $B$ to $T$ in the node containing $A$;
            If the $B$-$E$ connection is mandatory
                Designate $B$ in $T$ as a continuation attribute;
        Elseif $A \rightarrow B$
            Add $B$ to $T$ in the node containing $A$;
        Elseif $B \rightarrow A$
            Add $B$ to $T$ in a new child node of the node containing $A$;
            If the $B$-$E$ connection is mandatory
                Designate $B$ in $T$ as a continuation attribute;
        Else Add $B$ to $T$ in the node containing $A$;
```

Figure 4: Algorithm 1—Binary Algorithm

that Algorithm 1 can be used to generate only scheme-tree forests that have no potential redundancy with respect to functional and multivalued constraints.

Although Algorithm 1 can guarantee no potential redundancy, it does not guarantee that the scheme trees are as compact as possible. To get XNF (no potential redundancy *and* maximum compactness), we can add the following to Algorithm 1:

- Before the *Until* statement, add the following statements:

```
Compute the functional closure of each vertex using only fully functional edges
    (i.e. using only functional edges whose domain side is not optional);
Order the vertices with the first sort key being the number of closures in which the vertex appears
    (descending order), the second sort key being the closure size (descending order), and the third
    sort key being arbitrary (alphabetical in ascending order will do);
Discard from the tail-end of the ordered list, those vertices included in only a single closure;
Order the discarded vertices with the first sort key being the number of incident edges (descending
    order) and the second sort key being arbitrary (again, alphabetical in ascending order will do);
Append this list of ordered “discarded” vertices to the tail-end of the first ordered list
    (note that either one of the two ordered lists being joined together may be empty);
```

- Change the *If-Else* statement that selects the root node of a new scheme tree in Algorithm 1 to:

```
From the ordered list of vertices, select the first unmarked vertex $V$ in $H$ to be the root node
    of a new scheme tree $T$.
If there is no unmarked vertex left in the list, then select the marked vertex $V$ such that $V$ is contained
    in an unmarked edge and $V$ comes before any other vertices contained in unmarked edges in the list.
```

We call Algorithm 1 with this modification *Algorithm 1.1*.

Example 10 For the CM hypergraph in Figure 1(a), fully functional closures are: $Department^+ = \{Department\}$, $Faculty\ Member^+ = \{Faculty\ Member, Department\}$, $Grad\ Student^+ = \{Grad\ Student, Faculty\ Member, Department, Program\}$, $Program^+ = \{Program\}$, $Hobby^+ = \{Hobby\}$. Thus, *Department* is included in three closures, *Faculty Member* is included in two, *Grad Student*, *Program*, and *Hobby* are included in one with *Grad Student* having the largest closure size. Since the last three vertices on the list are included in only a single closure, they are ordered according to their respective number of incident edges: *Grad Student* has 3, *Hobby* has 2, and *Program* has 1. Hence, the order is *Department*, *Faculty Member*, *Grad Student*, *Hobby*, *Program*. Observe that for Algorithm 1.1, we produce the first scheme tree in Example 3. □

An alternate criteria for selecting the starting node for a new scheme tree that often gives intuitively better trees is:

```
Let $V$ be the selected vertex in Algorithm 1.1.
If $V$ has exactly one incident unmarked edge $E$ and $E$ is fully functional from $W$ to $V$, i.e. $W \rightarrow V$,
    select W, i.e. let $V$ be $W$ instead;
Else Select $V$;
```

We call Algorithm 1 with this modification *Algorithm 1.2*.

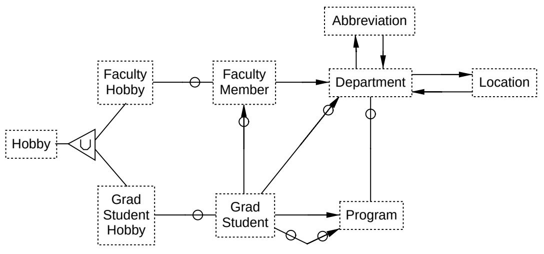
Example 11 Observe that Algorithm 1.2 produces the second scheme tree in Example 3. □

As we shall prove in the next section, we can use Algorithm 1 to guarantee no potential redundancy for any starting vertex in Figure 1(a). Further, the algorithm can guarantee the fewest scheme trees if the starting vertices for scheme trees are chosen according to one of the two criteria presented. Thus, we can use Algorithm 1.1 or Algorithm 1.2 to produce scheme trees in XNF.

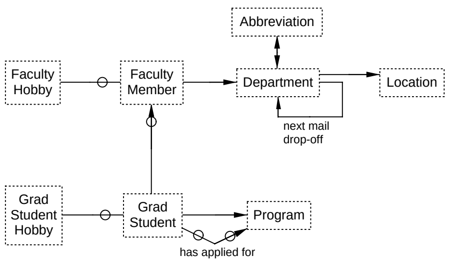
4 Assumptions, Requirements, and Guarantees

Unfortunately, Algorithms 1, 1.1, and 1.2 do not work for any CM hypergraph. Two conditions are required: (1) canonical and (2) binary. We discuss the canonical requirement in this section and show how to remove the binary requirement in the next section where we give a more general algorithm for producing scheme trees for XNF-compliant XML documents.

Definition 10 A binary CM hypergraph H is *canonical* if (1) no edge of H is redundant; (2) no vertex of H is redundant; and (3) bidirectional edges represent bijections. □



(a) Noncanonical Model Instance



(b) Canonical Model Instance

Figure 5: Noncanonical and Canonical Model Instances

Definition 11 Let E be an edge (V be a vertex) of a CM hypergraph H with a valid interpretation. Let H' be H without E (without V and all edges and generalization/specialization constraints incident on V). Edge E (vertex V) is *redundant* if (1) H' *preserves the information* in H (i.e. if there exists a procedure to construct H, including its data instance, from H')[12] and (2) H' *preserves the constraints* of H (i.e. if the constraints of H' imply the constraints of H). □

Example 12 Figure 5(a) shows a noncanonical CM hypergraph; Figure 5(a) shows its canonical counterpart. To illustrate the edge-redundancy requirement, consider the edge between *Grad Student* and *Department*. If it means the department of the student's faculty advisor, it is redundant.[13] Its removal preserves both information and constraints—we can recompute it as a join-project over the advisor and faculty/department relationship sets, and the constraints of these same two relationship sets imply both its functional and its mandatory and optional participation constraints. Similarly, if the edge between *Program* and *Department* represents the department that administers the student's program, it is redundant and can be removed. Assuming that the edge with optionals between *Grad Student* and *Program* represents a program a student has applied for, whereas the edge with no optionals represents the program in which the student is currently enrolled, neither edge is redundant. To illustrate the vertex-redundancy requirement, consider *Hobby*, which is a redundant object set as explained earlier in Example 7. To illustrate the bidirectional-edge requirement, consider the functional edges between *Department* and *Abbreviation* and between *Department* and *Location*. For the abbreviations, we have a bijection between departments and their abbreviations (e.g. *Computer Science* and *CS*), but for the locations we have a permutation—*Department* $\rightarrow$ *Location* gives the address of the department, whereas *Location* $\rightarrow$ *Department* gives the department for next mail drop for a mail carrier. Figure 5(b) shows a canonical CM hy-

[12]For vertex removal, we allow ourselves to know which edges and which generalization/specialization constraints we are expected to construct and populate.

[13]We make neither the universal-relation assumption nor the universal-relation-scheme assumption [FMU82, Ken81].

pergraph. No relationship set or object set is redundant, and the bidirectional edge[14] represents its only bijection. Since we have a permutation, we choose to model it as a permutation in a recursive relationship set Figure 5(b) shows.[15] □

Using the model instances in Figure 5(a), we can illustrate the redundancy problems that can arise from Algorithm 1 when a CM hypergraph is not canonical.

Example 13 Starting with *Department* in Figure 5(a), Algorithm 1 generates the scheme-tree forest with trees *(Hobby)** and *(Department, Abbreviation, (Abbreviation)*, Location, (Location)*, (Faculty Member, (Faculty Hobby)*, (Grad Student, (Grad Student Hobby)*, Program, Program)*)*, (Grad Student)*, (Program)*)**. First observe that *(Hobby)** is redundant as explained in Example 7. Next observe that *(Program)** is a list of programs for a department, but according to the model, these can all be computed by finding the programs of the grad students being advised by faculty members in the department. Similarly, *(Grad Student)** is a list of grad students in a department, but these can all be computed by finding the grad students being advised by faculty members in the department. Finally, observe that the constructions *Abbreviation, (Abbreviation)** and *Location, (Location)** are strange. For the departmental abbreviations which are in a one-to-one correspondence with the departments, we should drop the second mention in *(Abbreviation)** because it is redundant. For the locations, *(Location)** represents the address of the department whereas *Location* represents the address of the department next on the list in a mail route; thus, they should both be left as they are. As we shall see in the next example, however, we can fix this awkward construction by a heuristic modification. □

Example 14 By way of contrast to Example 13, starting with *Department* in Figure 5(b),[16] Algorithm 1 generates the scheme tree forest *(Department, Abbreviation, Location, Department of next mail drop-off, (Faculty Member, (Faculty Hobby)*, (Grad Student, (Grad Student Hobby)*, Program,, Program applied for)*)*)**. Observe that the redundant lists of programs, grad students, and hobbies are not present and that the redundant abbreviations and awkward locations have disappeared. □

We now turn our attention to proving that Algorithm's 1.1 and 1.2 generate XNF scheme trees.

Lemma 1 Let T be a properly constructed scheme tree for a CM hypergraph M. Let P be a path of T and let E be an edge in M that is contained in P. Let N_E be the closest node to the root node in P such that $E \subseteq Ancestor(N_E)$. T has potential redundancy with respect to a multivalued

[14]Note that $A \leftrightarrow B$ is not the same as $A \rightarrow B$ and $A \leftarrow B$. The former indicates a bijection, whereas the latter simply means two functional relationships between A and B.

[15]This choice does not effect our XNF result, but it is a heuristic transformation that almost always produces a more pleasing result.

[16]For clarity we make use of the names designating the meaning of some of the relationship sets. We suggest the use of clarifying information whenever there may be ambiguity, or even just whenever additional clarifying explanations are appropriate or desired.

constraint X—Y, where $\{X, Y\}$ is a partition of E, if and only if $E \not\rightarrow \mathit{Ancestor}(N_E)$.
Proof Sketch: The main idea[17] is that if $E \not\rightarrow \mathit{Ancestor}(N_E)$, the nesting of T cannot collapse all of the data values in an instance over E. On the other hand, if $E \rightarrow \mathit{Ancestor}(N_E)$, then the nesting is able to collapse all of the data values. $\square$

Lemma 2 Let H be a canonical CM hypergraph with no explicit generalization/specialization whose edges are all binary. Let T be a scheme tree generated by Algorithm 1. Let A be a continuation attribute in T, and let N_A be the node in T such that $A \in N_A$. $A \rightarrow \mathit{Ancestor}(N_A)$.
Proof Sketch: This lemma can be proved by induction on the number of vertices contained in T and by analyzing the four cases in Algorithm 1. $\square$

Theorem 1 Let H be a canonical CM hypergraph with no explicit generalization/specialization whose edges are all binary. Let T be a scheme tree generated by Algorithm 1. T has neither potential redundancy with respect to multivalued constraints nor potential redundancy with respect to functional constraints.
Proof Sketch: We proceed by induction on the number of iterations of the while-loop in constructing T. Using Lemmas 1 and 2, the result follows. $\square$

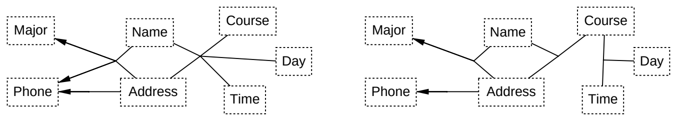
We now prove that Algorithm 1.1 is able to generate minimal number of scheme trees from the given binary canonical CM hypergraph H. However, we first need to define full FD paths in H, which are useful in the proof. (1) A vertex V in H is a *(trivial) full FD path* from V to V. (2) If $V_1 \rightarrow V_2 \rightarrow \cdots \rightarrow V_n$, $n \geq 1$, is a full FD path from V_1 to V_n, and $V_n \rightarrow V_{n+1}$ is a functional edge in H with a mandatory connection on V_n, then $V_1 \rightarrow V_2 \rightarrow \cdots \rightarrow V_n \rightarrow V_{n+1}$ is a *full FD path* from V_1 to V_{n+1}. (3) The construction of a *full FD path* must follow Rules (1) and (2).

Lemma 3 Let H be a canonical CM hypergraph with no explicit generalization/specialization whose edges are all binary. Two vertices V_1 and V_2 (not necessarily distinct), where V_1 is in edge E_1 and V_2 is in edge E_2, can appear in the same scheme tree T with no potential redundancy with respect to either E_1 or E_2 if and only if there is a vertex V_3 in E_1, there is a vertex V_4 in E_2, and there is a vertex V_5 in H such that there is a full FD path from V_3 to V_5 in H, and there is a full FD path from V_4 to V_5 in H.
Proof Sketch: The if-part is obvious since we may simply choose V_5 as the root node of a new scheme tree T in Algorithm 1. The only-if part follows from Lemma 1. $\square$

Theorem 2 Let H be a canonical CM hypergraph with no explicit generalization/specialization whose edges are all binary. If F is a scheme-tree forest generated from H by Algorithm 1.1, then each tree in F is in $\mathrm{XNF}_{\{\mathrm{FC,MC}\}}$.[18]

[17]Full proofs for all proof sketches can be found in a technical report, which is a longer (unpublished/unsubmitted) version of this paper, at http://osm.cs.byu.edu/Papers.html.

[18]In this notation, FC denotes a functional constraint, MC denotes a multivalued constraint, and IC denotes an inclusion constraint.



(a) N-ary Noncanonical Model Instance

(b) N-ary Canonical Model Instance

Figure 6: N-ary Noncanonical and Canonical Model Instances

Proof Sketch: We show, using Lemma 3, that generating a scheme tree will not hinder the clustering of two vertices that can be clustered together. We then analyze the four cases on how two vertices can be clustered, and the result follows. $\square$

Corollary 1 Let H be a canonical CM hypergraph with no explicit generalization/specialization whose edges are all binary. If F is a scheme-tree forest generated from H by Algorithm 1.2, then each tree in F is in $\text{XNF}_{\{\text{FC,MC}\}}$.
Proof Sketch: Since W is designated as a continuation attribute whether V or W is selected as the root node, the extent of the scheme tree does not change. $\square$

5 N-ary Algorithm

In this section we generalize Algorithm 1 for CM hypergraphs with n-ary edges. Three new problems arise in the generalization: (1) n-ary edges may be compositions of edges with lesser arity, (2) connecting sets of attributes between relationship sets may have different meanings, and (3) there are more degrees of freedom for scheme-tree configurations. We discuss each of these problems in turn.

Edge Decomposition. As one example of edge decomposition, consider the edge connecting *Name*, *Address*, *Phone*, and *Major* in Figure 6(a). If the phone for the person identified by the *Name-Address* pair is the home phone at the address of the person (i.e. is not the cell phone, for example), the 4-ary edge can be reduced by making the phone dependent only on the address as Figure 6(b) shows. As another example of edge decomposition, consider the 5-ary edge in Figure 6(a). Assuming that the schedule depends only on the course itself (the normal assumption unless the course is an individual-instruction course), we can decompose the edge as Figure 6(b) shows. Whether we can split the day from the time depends on the scheduling policy; our choice in Figure 6(b) assumes that courses can be scheduled at different times on different days.

There are multiple ways an edge can be decomposed, but these are all found in the relational database literature. Thus, we only mention them here without redeveloping them. Included are reductions to satisfy the requirements of 3NF (head and tail reductions in [Emb98]), reductions to satisfy the stronger requirements of BCNF (embedded FD reductions [Emb98]), and reductions to satisfy the even stronger requirements of 4NF and 5NF (non-FD edge reductions [Emb98]). The sample reduction for home phones above is a head reduction (right-hand-side reduction in [Mai83]), and the sample reduction for course schedules is a non-FD edge reduction (a lossless-join reduction in [Mai83]).

To accommodate these requirements, we now augment our definition of what it means for a CM hypergraph to be canonical.

Definition 12 A binary CM hypergraph H is *canonical* if (1) no edge of H is redundant; (2) no vertex of H is redundant; (3) bidirectional edges represent bijections; and (4) every n-ary edge is fully reduced. □

Example 15 The CM hypergraph in Figure 6(a) is noncanonical—neither the 4-ary nor the 5-ary edge is fully reduced. Assuming, as stated earlier, that the *Course-Day-Time* relationship set cannot be further reduced, the CM hypergraph in Figure 6(b) is canonical. □

Different Meanings. Consider *Name-Address* pairs in Figure 6(b). Suppose there are two names n_1 and n_2 and two addresses a_1 and a_2. Further suppose that in the relationship set with $Major$, n_1 relates to a_1 and n_2 relates to a_2, but in the relationship set with *Course*, n_1 relates to a_2 and n_2 relates to a_1. Under these assumptions, we cannot have the scheme tree *(Major, (Name, Address, (Course)*)*)**, which we would expect should be permissible. We cannot have this scheme tree because under *Major* our scheme-tree instance would have the subtuples $\{(Name,\ n_1),\ (Address,\ a_1)\}$ and $\{(Name,\ n_2),\ (Address,\ a_2)\}$, but in order to nest courses under these tuples, we need $\{(Name,\ n_1),\ (Address,\ a_2)\}$ and $\{(Name,\ n_2),\ (Address,\ a_1)\}$.

In general, to provide for nesting, the projections on the intersecting attribute sets between two edges must be identical for every valid interpretation. This condition holds when the projections on the intersection object sets between two edges have the "*same meaning.*" Indeed, for the CM hypergraph in Figure 6(b), $Name\text{-}Address$ pairs have the same meaning in both the $Major$ relationship set and the $Course$ relationship set—in both, the pair identifies an individual student.

Degrees of Freedom. Consider the ternary is-taking-course relationship set in the CM hypergraph in Figure 6(b). The scheme trees we may create for this relationship set include, for example, (a) *(Course, Name, Address)**, (b) *(Address, (Name, Course)*)**, and (c) *(Name, (Address, (Course)*)*)**. Whereas there are only three possible scheme trees for a nonfunctional binary relationship set, there are thirteen possible scheme trees for a nonfunctional ternary relationship set such as the is-taking-course relationship set in Figure 6(b). For an n-ary edge in general, we may have any of the $2^n - 1$ sets of vertices in the root node of its scheme tree, any of the $2^{n_1} - 1$ sets of vertices of the remaining n_1 vertices not chosen for the root in its first sublevel node, any of the $2^{n_2} - 1$ sets of vertices of the remaining n_2 vertices not chosen for the root or first sublevel node in its second sublevel node, and so forth.

```
$Input$: a canonical CM hypergraph $H$ (with no explicit generalization/specialization).
$Output$: a scheme-tree forest.
Until all edges and vertices have been marked
    If one or more unmarked edges remain
        Select an unmarked edge $E$ in $H$;
        Create a single-path scheme tree $T$ from $E$ such that
            the set of nodes in $T$ is a partition of $E$, and
            either each vertex in the root node of $T$ is mandatory for $E$,
            or there is at most one vertex in the root node of $T$;
        For each vertex $V$ in $E$
            If the $V$-$E$ connection is mandatory or $V$ is the root node of $T$,
                Mark $V$ in $H$ and designate $V$ in $T$ as a continuation attribute;
        While there is an unmarked edge $E$ in $H$ such that
                (1) $P$ is a path in $T$,
                (2) $D$ is the maximal nonempty set of attributes in $E \cap P$ that
                    has the "same meaning" in both $E$ and $P$,
                (3) each of the attributes in $D$ is designated as a continuation attribute in $P$,
                (4) there exists a node $N$ in $P$ such that $D \subseteq Ancestor(N)$ and
                    $E \rightarrow Ancestor(N)$;
                (5) If there are more than one nodes that satisfy Condition 4,
                    let $N$ be the lowest one.
            Mark $E$ in $H$;
            Create a single-path scheme tree $T'$ from $E - D$ such that
                the set of nodes in $T'$ is a partition of $E - D$;
            Make the root node of $T'$ the child of $N$;
            For each vertex $V$ in $E$
                If the $V$-$E$ connection is mandatory,
                    Mark $V$ in $H$ and designate $V$ in $T$ as a continuation attribute;
    Else
        Select an unmarked vertex $U$;
        Make $U$ the root node of a new scheme tree;
        Mark $U$ in $H$;
```

Figure 7: Algorithm 2—N-ary Algorithm

The additional degrees of freedom make it more difficult to specify a scheme-tree generation algorithm, but the idea for the generalization of Algorithm 1 is straightforward—we are still searching for the largest hierarchical structures embedded within the CM hypergraph. Figure 7 shows Algorithm 2, which generalizes Algorithm 1 by allowing n-ary edges.

Example 16 Consider the model instance in Figure 6(b) as the input to Algorithm 2. We choose the initial single-path scheme tree to be *Major* as the root, with *Name* and *Address* together in a child node of the root. We mark all three attributes in the given hypergraph and designate all of them as continuation attributes in the scheme tree. Since the edge $\{Name, Address, Course\}$ satisfies the five conditions of the while-loop, we add *Course* as a child of the node containing *Name* and *Address*. *Course* is then marked and designated as a continuation attribute. However, we can attach neither the edge $\{Course, Day, Time\}$ nor the edge $Address \rightarrow Phone$ to the scheme tree since we cannot find the node N in Algorithm 2 that satisfies the five conditions for these two edges. In particular, Condition 4 cannot be satisfied. We thus create a scheme tree for each of these two edges. The resulting scheme-tree forest is: *(Major, (Name, Address, (Course)*)*)**, *(Course, (Day, Time)*)**, *(Address, Phone)**. □

Although still an open question for future research, there appears to be no effective algorithm for guaranteeing the fewest possible scheme trees. The problem is that there are too many degrees of freedom—too many ways to add an n-ary edge to a scheme tree. Unfortunately, the choice makes a difference and the proper choice cannot always be determined until additional edges are added to a scheme tree. Thus, backtracking is required.

Example 17 Consider a hypergraph with three edges AB, ACD, and ACE. The only scheme tree that contains all three edges is the one with A as the root node, B and C as the children of A, and D and E as the children of C. Thus for three edges, we have three intersections to check, each of which is the intersection of two edges. In general, if there are n edges in the given hypergraph, there are $C(n, 2)$ intersections to check (assuming they all have the same meaning in the intersections). Obviously, this ordering of intersections takes exponential time.

We can nevertheless apply variations to Algorithm 2 that are similar to the variations of Algorithm 1, as specified in Algorithm 1.1 and 1.2. For CM hypergraphs whose edges only intersect on single object sets or whose only multiple-object-set edge intersections follow a single path in the hypergraph, we can use variations similar to Algorithm 1.1 and 1.2 to guarantee minimality. Since real-world CM hypergraphs tend to have mostly binary edges or tend to satisfy these stricted constraints, there is usually an effective algorithm for generating XNF. Furthermore, when these conditions do not hold, the subgraphs over which nondeterministic backtracking must be applied is usually small enough to allow for an exhaustive search. We leave for future research precise characterizations of these claims.

In the meantime, for this paper, we let Algorithms 2.1 and 2.2 be similar in spirit to Algorithms 1.1 and 1.2,[19] and we let Algorithm 2.3 be Algorithm 2 altered (1) to generate nondeterministic threads for all possible node configurations when an edge is added to a scheme tree and (2) to select a final minimal scheme-tree forest from the ones nondeterministically generated as a final step in the algorithm.

Theorem 3 Let H be a canonical CM hypergraph with no explicit generalization/specialization. Let T be a scheme tree generated by Algorithm 2. T has neither redundancy with respect to a multivalued constraint nor redundancy with respect to a functional constraint.
Proof Sketch: The proof is similar to the proof of Theorem 1. □

Corollary 2 Let H be a canonical CM hypergraph with no explicit generalization/specialization. If F is a scheme-tree forest generated from H by Algorithm 2.3, then each tree in F is in $\text{XNF}_{\{\text{FC,MC}\}}$.

[19]We do not specify these exactly since we cannot prove that they lead to XNF scheme-tree forests.

```
Input: a canonical CM hypergraph H.
Output: a scheme-tree forest.
"Collapse" each generalization/specialization hierarchy;
If the edges are all binary, Execute Algorithm 1.1 or 1.2;
Else Execute Algorithm 2.3;
```

Figure 8: Algorithm 3—General Algorithm

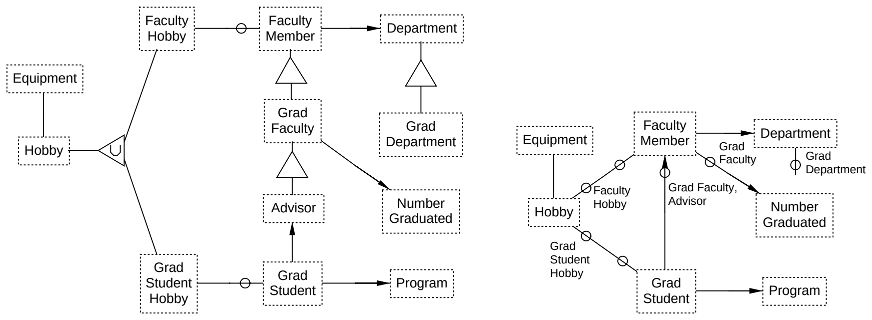
Proof Sketch: Theorem 3 guarantees no potential redundancy and the last step of Algorithm 2.3 guarantees that the generated scheme-tree forest has a minimal number of scheme trees. □

6 General Algorithm

In this section we further generalize our algorithms to allow CM hypergraphs with explicit generalization/specialization denoted by *ISA* triangles in CM hypergraphs. This generalization causes two new problems: (1) object sets may be redundant, and (2) ISA constructs must be considered in scheme-tree generation algorithms. We discussed and illustrated object-set redundancy in Section 2: whenever we can compute the contents of an isolated root generalization object set as explained in Example 7, we eliminate it. To handle all other ISA constructs, we collapse them into roles and thus eliminate them too. Once all ISA constructs have been eliminated, we are left with CM hypergraphs that can be processed by Algorithm 2, or by Algorithm 1 if all relationship sets happen to be binary. Figure 8 gives Algorithm 3.

We collapse ISA constructs as follows. We first preprocess any ISA triangles having n specializations, $n > 1$, by splitting the ISA into n ISA constructs, one for each specialization. We then (1) discard the specialization object set and the ISA construct, (2) attach all relationship sets incident on the specialization object set to the generalization object set and make all the connections optional, and (3) add the name of the discarded object set as a role name for the connection. When we collapse a chain of ISA constructs, we collect all the role names, separating them by commas.

Example 18 Figure 9 illustrates the process of collapsing ISA constructs and generating roles. We split the *Hobby* ISA construct in Figure 9(a) into two ISA constructs and then discard both specializations and make *Faculty Hobby* and *Grad Student Hobby* roles on optional connections for relationship sets respectively between *Hobby* and *Faculty Member* and *Hobby* and *Grad Student* as Figure 9(b) shows. For the ISA hierarchy under *Faculty Member* in Figure 9(a), we first make *Advisor* a role of *Grad Faculty* and then make *Grad Faculty* a role of *Faculty Member*. In addition, *Number Graduated* is now optionally connected to *Faculty Member* as Figure 9(b) shows. When there are no incident relationship sets, we simply discard the specialization and leave the role on an optional connection to the generalization, as we have done for *Grad Department* in Figure 9(b). □



(a) Model Instance with Explicit Generalization/Specialization

(b) Transformed Model Instance with no Explicit Generalization/Specialization

Figure 9: Model Instances with and without Explicit Generalization/Specialization

Example 19 Given the CM hypergraph in Figure 9(b), we can use Algorithm 1.2 to generate the scheme-tree forest: *(Department, (Faculty Member, Number Graduated, (Hobby)*, (Grad Student, Program, (Hobby)*)*)*)*, (Hobby, (Equipment)*)** □

Theorem 4 Let H be a canonical CM hypergraph. If F is a scheme-tree forest generated from H by Algorithm 3, then each tree in F is in $\mathrm{XNF}_{\{\mathrm{FC,MC,IC}\}}$.
Proof Sketch: Algorithm 3 requires H to be canonical, collapses all generalization/specialization hierarchies, and then uses an algorithm known to produce XNF. Thus, each tree generated is in $\mathrm{XNF}_{\{\mathrm{FC,MC,IC}\}}$. □

Although we have shown that scheme-tree forests generated by Algorithm 3 are in XNF, we nevertheless observe that all the specialization information is implicit. In the scheme-tree forest in Example 19, for example, there is no explicit way to say that a faculty member is an advisor. Implicitly, we can say that the faculty members who associate with at least one grad student are advisors, but this inference is only implicit.

To solve this problem, we can augment our scheme trees by adding roles. We add a question mark to each role name, and then place these augmented names in parentheses, and append the parenthesized names to an object set. We place roles in scheme trees as we encounter the connections to which they belong in our scheme-tree-generation algorithms.

Example 20 Augmented with role names the scheme trees in Example 19 become: *(Department (Grad Department?), (Faculty Member, (Grad Faculty?, Advisor?), Number Graduated, (Hobby (Faculty Hobby?))*, (Grad Student, Program, (Hobby, (Grad Student Hobby?))*)*)*)*, (Hobby, (Equipment)*)** □

We use role names in two ways when we generate XML DTDs: (1) we use them in place of an object set, and (2) we add them as attributes. If the object set to which a role is attached is a continuation attribute in the scheme tree, we add the role as an attribute in the object set's XML tag. If, on the other hand, the object set to which the role is attached is not a continuation attribute, we replace the tag name with the tag name of the role.

Example 21 Figure 10(a) shows an example of the DTD we can generate from the scheme trees in Example 20, and Figure 10(b) shows a partial possible complying XML document. Note that *Faculty Hobby* and *Grad Student Hobby* are XML tags (because each is associated with a noncontinuation attribute), whereas *Grad Faculty*, *Advisor*, and *Grad Department* are XML attributes (because each is associated with a continuation attribute). □

7 Concluding Remarks

We have proposed and formally defined XNF (XML Normal Form). XNF guarantees that complying XML documents have no redundant data values and have maximally compact connectivity. We have also developed conceptual-model-based algorithms (Algorithms 1.1, 1.2, 2.3, and 3) to generate DTDs to ensure that complying documents are in XNF, and we have proved that these algorithms are correct (Theorems 1–4 along with Corollaries 1–3).

We have implemented a tool to work synergistically with a user to develop XNF-compliant DTDs. Currently our tool allows users to specify CM diagrams, to convert diagrams to CM hypergraphs, to apply either Algorithms 1.1 or 1.2, or to select root nodes for scheme trees and then apply Algorithm 1. In another tool, we have implemented, users can synergistically convert CM hypergraphs into canonical hypergraphs.

As for current and future work, we need to integrate these two tools so that we can have the synergistic system we desire. We also need to implement Algorithms 2 and 3 and their variations, and we need to formally characterize the restrictions to n-ary CM hypergraphs for which there exist effective scheme-tree generation algorithms. We also intend to investigate additional heuristic variations of scheme-tree generation algorithms and synergistic development of XNF-compliant DTDs.

Acknowledgements: This material is based upon work supported by the National Science Foundation under grant No. IIS-0083127. Kimball Hewett programmed the interface we used for our implementation, Eric Carter implemented the canonicalization procedures, and Troy Walker coded the implemented algorithms described in this paper.

References

[AD93] P. Atzeni and V. DeAntonellis. *Relational Database Theory.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1993.

[BGH00] L. Bird, A. Goodchild, and T. Halpin. Object role modelling and xml-schema. In *Proceedings of the Ninteenth International Conference on Conceptual Modeling (ER2000)*, pages 309–322, Salt Lake City, Utah, October 2000.

[BR00] V. Bisová and K. Richta. Transformation of uml models into xml. In *Proceedings the 2000 ADBIS-DASFAA Symposium on Advances in Databases and Information Systems*, pages 33–45, Prague, Czech Republic, September 2000.

[CSF00] R. Conrad, Deiter Scheffner, and J.C. Freytag. XML conceptual modeling using UML. In *Proceedings of the Ninteenth International Conference on Conceptual Modeling (ER2000)*, Salt Lake City, Utah, October 2000. 558–571.

[EKW92] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-oriented Systems Analysis: A Model-Driven Approach.* Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[Emb98] D.W. Embley. *Object Database Development: Concepts and Principles.* Addison-Wesley, Reading, Massachusetts, 1998.

[FMU82] R. Fagin, A.O. Mendelzon, and J.D. Ullman. A simplified universal relation assumption and its properties. *ACM Transactions on Database Systems*, 7(3):343–360, September 1982.

[Hal99] T. Halpin. *Conceptual Schema & Relational Database Design.* WytLytPub, revised 2nd edition, 1999.

[Ken81] W. Kent. Consequences of assuming a universal relation. *ACM Transactions on Database Systems*, 6(4):539–556, December 1981.

[Mai83] D. Maier. *The Theory of Relational Databases.* Computer Science Press, Inc., Rockville, Maryland, 1983.

[MEN96] W.Y. Mok, D.W. Embley, and Y-K. Ng. A normal form for precisely characterizing redundancy in nested relations. *ACM Transactions on Database Systems*, 21(1):77–106, March 1996.

[Mok] W.Y. Mok. A comparative study of various nested normal forms. *IEEE Transactions on Knowledge and Data Engineering.* (to appear).

[XMI] Home Page for OASIS's XML Metadata Interchange (XMI). URL: http://www.oasis-open.org/cover/xmi.html.

[XML] Home Page for a listing of organizations producing industry-specific XML DTDs. URL: http://xml.org/xmlorg_registry/index.html.

[XML00] XML schema part 0: Primer: W3?c working draft, 2000. URL: http://www.w3.org/TR/2000/WD-xmlschema-0-20000407/.

```
<!DOCTYPE University[
    <!ELEMENT University (
        (Department, (Faculty_Member,
            Number_Graduated, (Faculty_Hobby)*,
            (Grad Student, Program,
            (Grad_Student_Hobby)* )* )* )*,
        (Hobby, (Equipment)* )* )>
    <!ELEMENT Department (#PCDATA)>
    <!ATTLIST Department
        Role CDATA #IMPLIED>
    <!ELEMENT Faculty_Member (#PCDATA)>
    <!ATTLIST Faculty_Member
        Role CDATA #IMPLIED>
    <!ELEMENT Number_Graduated (#PCDATA)>
    <!ELEMENT Faculty_Hobby (#PCDATA)>
    <!ELEMENT Grad_Student (#PCDATA)>
    <!ELEMENT Program (#PCDATA)>
    <!ELEMENT Grad_Student_Hobby (#PCDATA)>
    <!ELEMENT Hobby (#PCDATA)>
    <!ELEMENT Equipment (#PCDATA)>
]>
```

(a) Sample DTD Specification Generated from an Augmented Scheme-Tree Forest

```
<University>
    <Department Role="Grad_Department">
            Math</Department>
        <Faculty_Member Role="Grad_Faculty & Advisor">
                Kelly</Faculty_Member>
            <Number_Graduated>23</Number_Graduated>
            <Faculty_Hobby>Chess</Faculty_Hobby>
            <Faculty_Hobby>Skiing</Faculty_Hobby>
            <Grad_Student>Pat</Grad_Student>
                <Program>PhD</Program>
                <Grad_Student_Hobby>
                    Hiking</Grad_Student_Hobby>
                <Grad_Student_Hobby>
                    Skiing</Grad_Student_Hobby>
            <Grad_Student>Tracy</Grad_Student>
                <Program>MS</Program>
                <Grad_Student_Hobby>
                    Hiking</Grad_Student_Hobby>
                <Grad_Student_Hobby>
                    Sailing</Grad_Student_Hobby>
        ...
        <Faculty_Member Role="Grad_Faculty">
                Noel</Faculty_Member>
        ...
        <Faculty_Member>Lynn</Faculty_Member>
    ...
    <Department>History</Department>
...
</University>
```

(b) Sample XML Document for the Augmented Specification

Figure 10: Sample DTD Generated from an Augmented Scheme-Tree Forest Along with a Sample Complying XML Document