Generating the Fewest Redundancy-Free Scheme Trees from Acyclic Conceptual-Model Hypergraphs in Polynomial Time

Wai Yin Mok[*], Joseph Fong[†] and David W. Embley[‡]

Abstract

Generating the fewest redundancy-free scheme trees from conceptual-model hypergraphs is NP-hard [11]. We show, however, that the problem has a polynomial-time solution if the conceptual-model hypergraph is acyclic. We define conceptual-model hypergraphs, cycles, and scheme trees, and then present a polynomial-time algorithm and show that it generates the fewest redundancy-free scheme trees. As a practical application for the algorithm, we comment on its use for the design of "good" XML schemas for data storage.

Keywords: Conceptual-model-based scheme-tree generation, data redundancy in scheme trees, minimal scheme-tree forests, acyclic conceptual-model hypergraphs.

1 Introduction

Generating schemas for data storage from conceptual models has a long-standing tradition. Its advantages are clear: (1) understandability, allowing both customer and developer to communicate effectively about the data to be included and the constraints to be enforced and (2) formality, allowing algorithmic derivation of schemas with good properties regarding the space needed to store the data and the time needed to query and update the data.

Following this conceptual-model tradition, we seek for algorithms to derive good schemas when the intended usage is for hierarchical data storage, such as in XML databases [3]. Like relational database schemas, we consider hierarchical scheme-tree storage structures

[*]Dept. of Economics and Information Systems, University of Alabama in Huntsville, Huntsville, Alabama 35899, USA, mokw@email.uah.edu.

[†]Dept. of Computer Science, City University of Hong Kong, Hong Kong, China, csjfong@cityu.edu.hk.

[‡]Dept. of Computer Science, Brigham Young University, Provo, Utah 84602, USA, embley@cs.byu.edu.

to be "good" when they prevent redundancy and are stored in as few schemas as possible. Preventing redundancy reduces storage and allows for simple constraint-satisfying update checks, thus reducing both space and time. Storing data in as few schemas as possible reduces query processing time when joins across populated schema instances are necessary.

We thus seek for algorithms that generate the fewest redundancy-free scheme trees from a conceptual-model hypergraph (a *CM hypergraph*). In [11] we give algorithms that generate redundancy-free scheme trees; but if the CM hypergraph has cycles, the algorithms cannot guarantee that the number of generated scheme trees will be minimal. Indeed, in [11] we prove that finding an algorithm for minimality in the general case is intractable, and we thus settle for providing a heuristic for finding the minimal number of scheme trees. Continuing this work, we consider placing restrictions on CM hypergraphs. We observe that if the universal-relation-scheme assumption (URSA) [14] holds for a CM hypergraph H, if H is Graham-reduction acyclic [10], and if each hyperedge in H is in BCNF [5], then we are able to find the largest redundancy-free scheme-tree storage structure in polynomial time [12]. Successively extracting the largest redundancy-free scheme tree from what remains of the CM hypergraph is a good heuristic for generating the fewest number of scheme trees. We continue our investigation here where we show that by using an alternate definition of hypergraph acyclicity that is strictly stronger than Graham reduction, a polynomial-time algorithm exists that guarantees the generation of the fewest number of redundancy-free scheme trees. Further, the algorithm needs neither the URSA nor the BCNF assumption.

By way of comparison with the scheme-tree normalization work of others (especially in the context of XML [1, 4, 9, 15, 16, 17, 18]), we point out that our approach differs significantly. Not only have these other researchers defined their FDs, and thus their normal forms, differently, the basis of our approach is also different from theirs. As opposed to constraints specialized for XML, which are defined in these papers, we rely on standard FD and hypergraph-generated MVD definitions—both of which can be straightforwardly derived from conceptual-model hypergraphs. Furthermore, the basis of our approach is conceptual models, which have not been considered at all in other XML normalization work. We believe our approach is more common in practice and in line with the tradition followed by information-system developers, who first create conceptual-model instances and then generate database storage structures.

To help clarify our intentions, we present some examples.[1] Example 1 gives an illustrative acyclic CM hypergraph along with some valid instance data. Examples 2 and 3 illustrate poor designs: respectively, a design with data redundancy and a fragmented design with more scheme trees than necessary. Example 4 illustrates a good design.

[1]We rely on intuition for some undefined terms in these introductory examples. We carefully define these terms later in the paper.

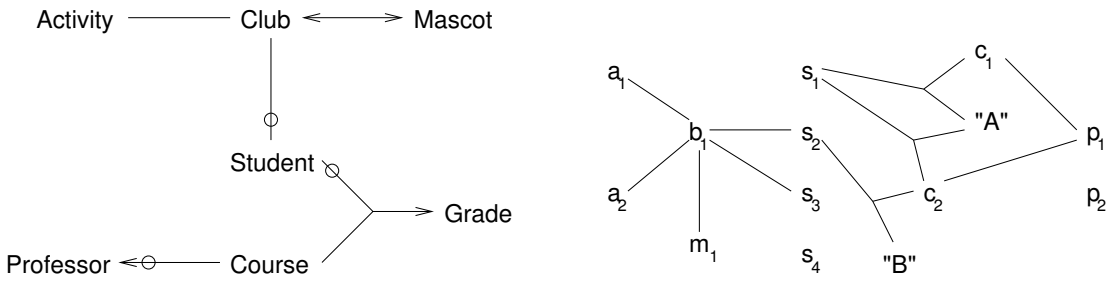

Figure 1: An acyclic CM hypergraph and a valid population of data.

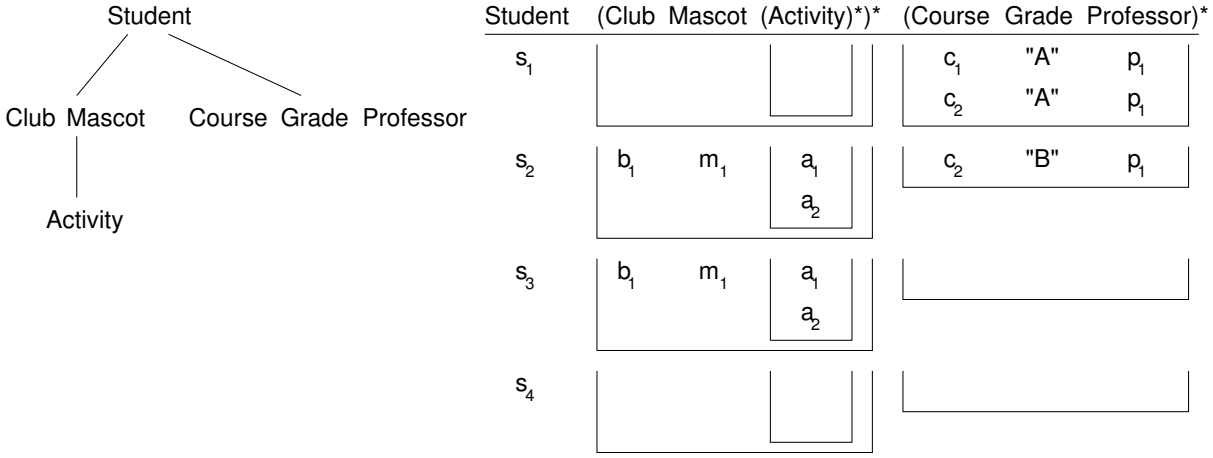

Figure 2: An incorrect design with data redundancy.

Example 1 Figure 1 shows an acyclic CM hypergraph H and a valid population of data for H. In the CM hypergraph, named vertices denote object sets. Edges are relationship sets among two or more object sets. Arrowheads denote functional relationship sets, and o's denote optional participation of objects in relationships. The data for H states that club b_1, whose members are students s_2 and s_3 and whose mascot is m_1, has activities a_1 and a_2. The data also states that professor p_1 teaches courses c_1 and c_2, but professor p_2 does not currently teach any course. And it states that student s_1 earned an A in both courses c_1 and c_2, student s_2 earned a B in course c_2, but students s_3 and s_4 are new students who have not yet earned a grade for any course, although s_3, but not s_4, has already joined a club. □

Example 2 Figure 2 shows a scheme tree for the hypergraph H in Figure 1 along with the result of populating it with the instance data in Figure 1. However, the scheme tree and its populated instance in Figure 2 are problematic. The FDs *Course* $\rightarrow$ *Professor*, *Club* $\rightarrow$ *Mascot*, *Mascot* $\rightarrow$ *Club* and the MVD *Club* $\twoheadrightarrow$ *Activity* are constraints implied by the acyclic CM hypergraph in Figure 1 that must hold. As a result, the populated scheme tree in Figure 2 has redundant data. Since course c_2 appears twice and *Course* $\rightarrow$ *Professor* holds, both appearances of c_2 must relate to professor p_1. Similarly, since club b_1 appears

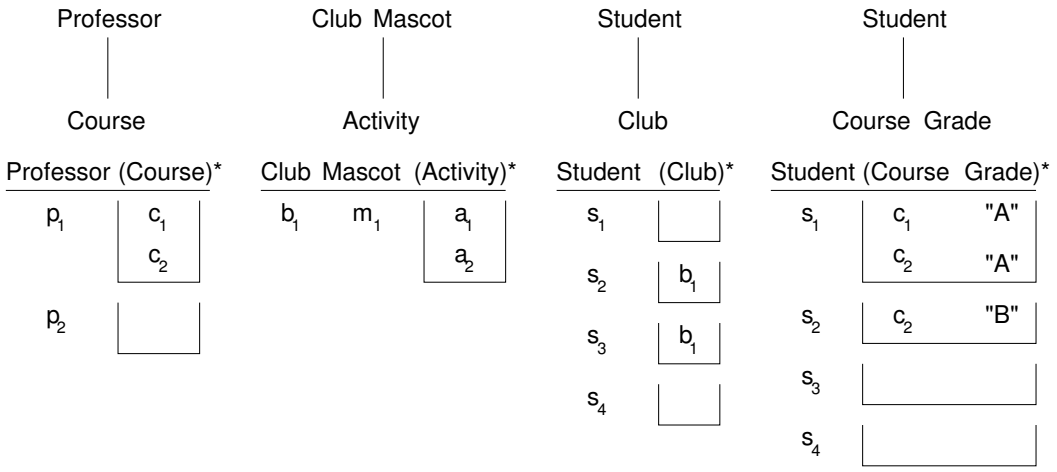

Figure 3: An incorrect design with unnecessary fragmentation.

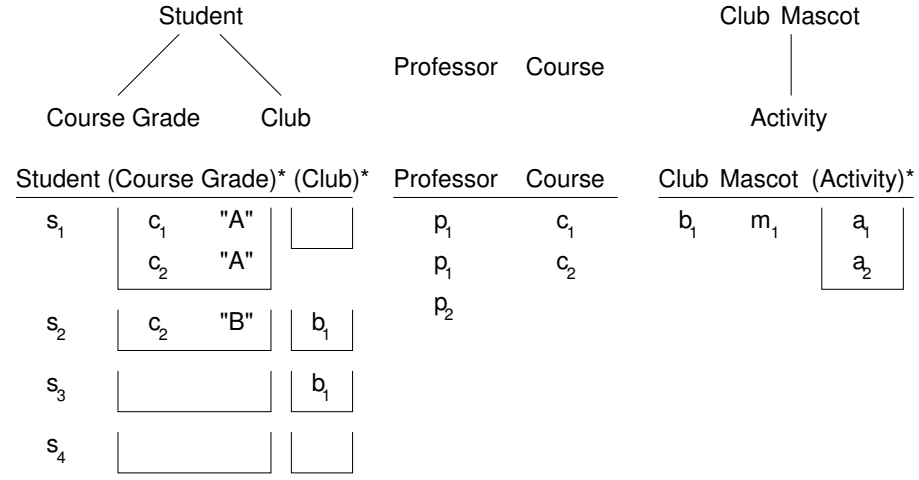

Figure 4: A correct design that avoids data redundancy and unnecessary fragmentation.

twice and $Club \to Mascot$ and $Club \twoheadrightarrow Activity$ hold, b_1's mascot and activities must appear twice, and since mascot m_1 appears twice and $Mascot \to Club$ holds, m_1's club b_1 must appear twice. In addition, professor p_2, who does not teach any course, cannot even be included in the populated scheme tree in Figure 2, which results in a loss of data. □

Example 3 Figure 3 shows another collection of scheme trees for the hypergraph H in Figure 1 along with the results of populating them with the data from Figure 1. While the scheme trees and their populated instances in Figure 3 do not have redundancy, they unnecessarily fragment the data. This means we have to combine the data from two or more populated scheme trees to answer some queries. For example, we have to combine the data from the last two populated scheme trees in Figure 3 to unite all the information directly related to students. This join turns out to be unnecessary, since they can be joined and stored without introducing redundancy. □

Example 4 In Figure 4 the scheme trees organize the data without causing redundancy and account for all the data in Figure 1. □

Contributions of the work reported in this paper include:

- Discovery of an appropriate definition of CM hypergraph acyclicity that enables a polynomial-time, algorithmic solution for generating the fewest possible redundancy-free scheme trees.

- Proofs of theorems guaranteeing that the algorithmic solution we propose generates a forest of redundancy free scheme trees (Theorem 1), that the generated forest of scheme trees collectively covers the information in a given CM hypergraph with the fewest number of scheme trees (Theorem 2), and that the algorithms in the solution all run in polynomial time (Theorem 3).

- Loosening of constraints from previous results allowing designers the freedom to apply CM hypergraphs in their analysis work without requiring that the URSA hold and without requiring that each hyperedge be in BCNF. The simplicity of the acyclicity requirement along with the results of the algorithmic solution provide designers with excellent guidelines for creating conceptual-model instances and mapping them into good designs for scheme-tree-based storage structures such as those found in XML.

We present the details of these contributions as follows. In Section 2 we define, illustrate, and explain basic terms. We rely on these fundamentals for explaining our scheme-generation algorithm (Section 3) and proving that it generates the fewest redundancy-free scheme trees in polynomial time (Section 4). Some of the proofs are lengthy and detailed; we therefore defer all proofs to the appendix, but do provide intuitive proof sketches for the three major theorems. In Section 5 we mention some practicalities about applying our algorithm for the design of XML storage structures, and we make concluding remarks in Section 6.

2 Fundamentals

2.1 CM Hypergraphs

We begin with definitions that tell us which hypergraph variant constitutes a conceptual-model hypergraph (CM hypergraph). We note, in particular, that the CM-hypergraph definitions provide for cardinality constraints typically found in database modeling: both functional and multi-valued constraints and both mandatory and optional participation constraints. We note also that the definitions allow several edges to connect among the same

vertices and also note that CM hypergraphs make neither the universal-relation assumption (URA) nor the universal-relation-scheme assumption (URSA) [8, 14].

Definition 1 A *hypergraph* is a pair (V, E), where V is a set of vertices and E is a set of edges. Each edge $E_i \in E$ is a multiset V_i $(1 \leq i \leq |E|)$ of at least two $(|V_i| \geq 2)$ vertices in V. (If $|V_i| = 2$, the edge is *binary*; if $|V_i| = 3$, the edge is *ternary*, ...; and, in general, if $|V_i| = n$, the edge is *n-ary*.) □

A hypergraph edge may have multiple connections (recursive connections) to a vertex. Thus, naming the vertex and the edge does not uniquely distinguish an edge-vertex connection. Hence, we introduce connection identifiers to distinguish them.

Definition 2 In a hypergraph (V, E), a unique *edge-vertex connection* C_i denotes each edge/vertex association $(1 \leq i \leq \sum_{j=1}^{|E|} |E_j|$ where E_j is an edge in $E)$. □

Definition 3 A *CM hypergraph* is a hypergraph with the following additional properties: (1) Each vertex V_i is an *object set*, whose elements denote objects of interest in the world being modeled. (2) Each edge E_j is a *relationship set*, whose elements denote relations over the objects in the object sets of the multiset of vertices of E_j. (3) Referential integrity holds; thus the projection of each edge E_j $(1 \leq j \leq |E|)$ on the object set S of an edge-vertex connection of E_j is a (not necessarily proper) subset of the objects in S. (4) Every edge E_j is undirected, uni-directed, or bi-directed. When E_j is directed, the multiset for E_j has two non-empty multisets, one for the tail(s) of the directed edge and one for the head(s). A uni-directed edge denotes a function from a cross-product of tail object set(s) to a cross-product of head object set(s), and a bi-directed edge, in addition, denotes an inverse function from head(s) to tail(s). (5) Every edge-vertex connection C_k for vertex V_i and edge E_j has either a "mandatory" or an "optional" declaration, which dictates whether the objects in V_i respectively must or may participate in the relationships in E_j. □

Example 5 In Figure 1, the CM hypergraph H has seven object sets, named *Activity*, *Club*, etc. H has four binary relationship sets and one ternary relationship set, {*Student*, *Course*, *Grade*}. This ternary edge is functional from *Student* × *Course* to *Grade*. All edge-vertex connections are mandatory except the three marked with the optional designator "o". These three optional connections allow for professors who do not teach courses, students who are not in clubs, and students who have not yet received grades for courses. □

2.2 Acyclic CM Hypergraphs

Since our results are for acyclic CM hypergraphs, we now present their definition. The literature gives several, non-equivalent definitions for cycles in hypergraphs. It makes a

difference which definition we use. Further, we show that, although CM hypergraphs require neither the URSA nor the URA, the definition of acyclic CM hypergraphs we use nevertheless allows us to make both the URSA and the URA. We eventually need both so that we can use standard relational dependency theory, which enables our redundancy-free, minimal-scheme-tree result. As we shall also see, however, designers need not concern themselves with making CM hypergraphs conform to the URA nor to the URSA beyond just making CM hypergraphs acyclic in the sense we define here. We sort out all these issues in the following definitions and lemmas.

Definition 4 A *path* in a CM hypergraph H is a sequence of the form V_1, C_1, E_1, C_2, V_2, $\ldots$, V_i, C_{2i-1}, E_i, C_{2i}, V_{i+1}, $\ldots$, V_n, C_{2n-1}, E_n, C_{2n}, V_{n+1}, where (1) $n \geq 1$, (2) V_1, $\ldots$, V_n are vertices of H, (3) E_1, $\ldots$, E_n are edges of H, (4) C_1, $\ldots$, C_{2n} are edge-vertex connections of H, and (5) each C_i in the sequence where i is odd conjoins its preceding vertex with its succeeding edge and where i is even conjoins its preceding edge with its succeeding vertex. A path is *simple* if its vertices are all distinct. □

Definition 5 A CM hypergraph H is *connected* if a path exists between every pair of distinct vertices of H. □

Without loss of generality, all CM hypergraphs considered in this paper are connected. For, if not, we may achieve the results we want by simply applying the algorithms in this paper to each individual connected component.

Definition 6 A *cycle* in a CM hypergraph H is a path in H with $V_1 = V_{n+1}$, where V_1 is the first vertex of the path and V_{n+1} is the last, and with every other vertex, edge, and edge-vertex connection unique. A CM hypergraph is *acyclic* if it does not have a cycle. □

Example 6 Figure 5 shows some cycles in CM hypergraphs.[2] The apparent cycle in Figure 5(a) satisfies Definition 4: *Professor*, *Instructor*, *is-teaching*, *OfferedCourse*, *Course*, *DeptCourse*, *administered-by*, *CourseDept*, *Department*, *DeptWithFaculty*, *has*, *FacultyMember*, *Professor*. Observe that for forming paths the direction of directional edges is immaterial—the cycle in Figure 5(a) proceeds "backwards" through the edge *Professor* → *Department*. In Figure 5(b), *Team*, *HomeTeam*, *plays*, *VisitingTeam*, *Team* is a cycle. Observe that paths traverse through binary parts of n-ary ($n > 2$) edges. In Figure 5(c), *Professor*, *Reviewer*, *reviews-for*, *AcademicJournal*, *Journal*, *ProfessionalJournal*, *published-by*, *Author*, *Professor* is one of several cycles. □

[2]Note that we have supplied names for every edge and for every edge-vertex connection as well as for every vertex. CM hypergraphs optionally allow edge and edge-vertex-connection names, which respectively are *relationship-set names* and *role names*. Making the names unique lets us unambiguously refer to an object set, relationship set, or role.

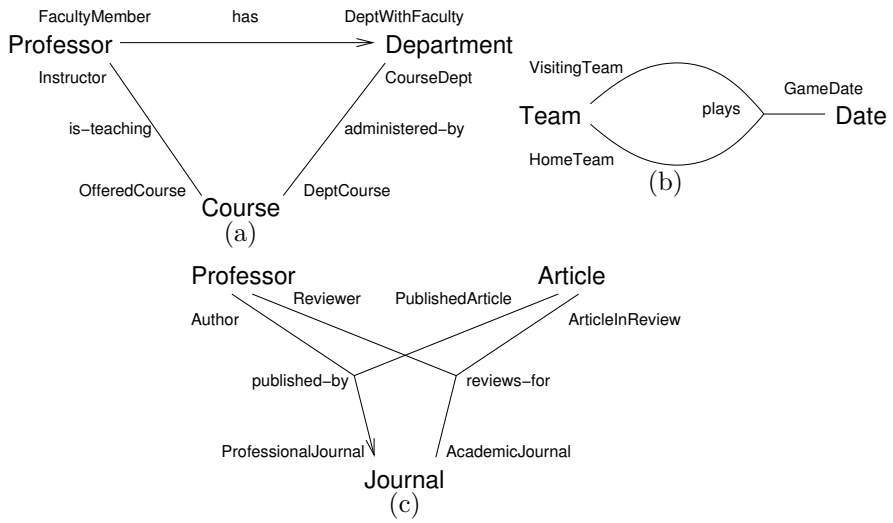

Figure 5: Different forms of cycles in CM hypergraphs.

Two fundamental differences between CM hypergraphs and the hypergraphs in Beeri, et al. [2] and Maier [10] prohibit us from taking direct advantage of their results. First, the edges in CM hypergraphs might be multisets of vertices whereas the edges in the hypergraphs of [2, 10] are sets instead. Second, the hypergraphs of [2, 10] make the URSA and often even the stronger URA whereas neither need hold in CM hypergraphs. Nevertheless, in the context of acyclic CM hypergraphs, these differences disappear. Lemma 4 and the lemmas leading to Lemma 4 show that acyclic CM hypergraphs satisfy the URSA. Further, Lemma 5 shows that, appropriately extended, populated CM-hypergraph instances with null placeholders satisfy the URA as well. Satisfying these assumptions allows us to make direct use of relational dependency theory. As a first immediate consequence, Lemma 8 shows that the edge-in-BCNF assumption always holds for any acyclic CM hypergraph.

Lemma 1 Let H be an acyclic CM hypergraph. Every edge in H is a set, not a multiset, of vertices.[3] □

Lemma 2 Let H be an acyclic CM hypergraph. If $E_i \cap E_j$ is not empty for distinct edges E_i and E_j in H, then $E_i \cap E_j$ contains exactly one vertex. □

Lemma 3 Let H be a connected acyclic CM hypergraph. Let V_1 and V_n be distinct vertices of H. There exists a unique simple path p in H from V_1 to V_n. □

Definition 7 Let U be a set of attributes. U satisfies the *universal-relation-scheme assumption* (*URSA*) if for any subset S of U, there is a unique relationship among the attributes in S [14]. □

[3]Proofs for lemmas and theorems are in the appendix.

Lemma 4 Let H be a connected acyclic CM hypergraph. The set of vertices in H satisfies the URSA. □

Although connected acyclic CM hypergraphs satisfy the URSA, their populated instances do not, in general, satisfy the stronger URA (the requirement that relations are all projections of a single universal relation). The presence of optionals on edge-vertex connections (or rather the absence of mandatory constraints) allows for dangling tuples which are lost in the join that creates the universal relation. We now show, however, that with the addition of nulls as placeholders for missing values, populated CM hypergraphs satisfy the URA.

Definition 8 Let $H = (V, E)$ be a connected acyclic CM hypergraph, and let H_D be a valid population of data for H[4]. Let r_i be the relation for edge E_i of H_D $(0 \leq i \leq |E|)$, and let s_i be the single-attribute relation for vertex V_i of H_D $(0 \leq i \leq |V|)$. A *universal relation* u of H_D for H is obtained as follows: (1) Let $r = r_1 \otimes r_2 \otimes \cdots \otimes r_{|E|}$ where for $1 \leq i < |E|$, $E_1 E_2 \ldots E_i \cap E_{i+1} \neq \emptyset$. (2) Let $u = r \otimes s_1 \otimes s_2 \otimes \cdots \otimes s_{|V|}$. □

When the schemas for r and s in $r \otimes s$ have no attribute in common, the outer join produces the cross product of r and s. Thus, the constraint $E_1 E_2 \ldots E_i \cap E_{i+1} \neq \emptyset$ for $1 \leq i < |E|$ is necessary to avoid generating connections among values not reflected in the data instance of an underlying CM hypergraph. Since our CM hypergraphs are connected, an ordering that satisfies this constraint always exists. Further, since we assume that nulls are distinguished, this ordering constraint ensures that the universal relation for a populated CM hypergraph is unique up to a renaming of nulls.

Example 7 Figure 6 gives the universal relation for the CM hypergraph and data instance in Figure 1. In the figure (and throughout the remainder of the paper), we let the attribute name A stand for *Activity*, B for *Club*, M for *Mascot*, S for *Student*, G for *Grade*, C for *Course*, and P for *Professor*. We compute the outer join as $AB \otimes BM \otimes BS \otimes SGC \otimes CP \otimes A \otimes B \otimes M \otimes S \otimes G \otimes C \otimes P$. The non-empty intersection constraint holds since $AB \cap BM = B, ABM \cap BS = B, ABMS \cap SGC = S$, etc. This constraint ensures, for example, that we do not compute the universal relation as $CP \otimes BS \otimes \cdots$, which would erroneously create a relationship between professor p_2 and student s_2.

Lemma 5 Let $H = (V, E)$ be a connected acyclic CM hypergraph, and let H_D be a valid population of data for H. Let u be the universal relation for H constructed from H_D in which the nulls are treated as distinguished values. If $E_1, \ldots, E_{|E|}$ be the edges of H, then u satisfies the join dependency $\bowtie(E_1, \ldots, E_{|E|})$. □

[4]A population of data for H is *valid* if it satisfies all the constraints of the CM hypergraph

A	B	M	S	C	G	P
a_1	b_1	m_1	s_2	c_2	"B"	p_1
a_1	b_1	m_1	s_3	⊥	⊥	⊥
a_2	b_1	m_1	s_2	c_2	"B"	p_1
a_2	b_1	m_1	s_3	⊥	⊥	⊥
⊥	⊥	⊥	s_1	c_1	"A"	p_1
⊥	⊥	⊥	s_1	c_2	"A"	p_1
⊥	⊥	⊥	s_4	⊥	⊥	⊥
⊥	⊥	⊥	⊥	⊥	⊥	p_2

Figure 6: Universal relation computed from the hypergraph- and data-instance in Figure 1.

Since the set of vertices in a connected acyclic CM hypergraph satisfies the URSA and its populated instance, appropriately extended with distinguished-null placeholders, satisfies the URA, we can apply standard dependency theory [10]. If H is a connected acyclic CM hypergraph, its *given functional dependencies* (FDs) and *multivalued dependencies* (MVDs) and its *given join dependency* (JD) are as follows:

- Each directed edge is a nontrivial FD.
- Each bi-directed edge yields two nontrivial FDs, one in each direction.
- The MVDs are hypergraph generated: Consider each single vertex A, one at a time. Remove A from every edge of H in which it appears. Let H_1, H_2, ..., H_n ($n \geq 1$) be the remaining disjoint connected sub-hypergraphs whose vertex sets are respectively Y_1, Y_2, ..., Y_n. If $n \geq 2$, then $A \twoheadrightarrow Y_1 \mid Y_2 \mid \cdots \mid Y_n$ are the nontrivial hypergraph-generated MVDs for A. (If $n = 1$, the generated MVD $A \twoheadrightarrow Y_1$ is trivial.) Also, since we will often wish to apply hypergraph-generated MVDs to vertex subsets, we note in Lemma 6 that for any set of vertices $V' \subseteq V$ (where V is the vertex set of H), if $A \in V'$, and $A \twoheadrightarrow Y$ is a hypergraph generated MVD, then $A \twoheadrightarrow Y \cap V'$ holds for V'.
- The given JD is $\bowtie(E_1, \ldots, E_n)$ where $E_1, \ldots, E_n$ are the edges of the hypergraph.

For our work here, we assume that the only FDs, MVDs, and JDs for a connected acyclic CM hypergraph are its given FDs, MVDs, and JD. Thus, all given dependences of interest are apparent from the hypergraph itself. Further, as Lemma 7 shows, we need only be concerned with the given FDs and MVDs since the given JD is equivalent to the given MVDs.

Lemma 6 Let $H = (V, E)$ be a connected acyclic CM hypergraph. Let V' be any subset of V. If $A \twoheadrightarrow S$ is a hypergraph-generated MVD from H and $A \in V'$, then $A \twoheadrightarrow S \cap V'$ holds on V'. □

Lemma 7 Let $H = (V, E)$ be a connected acyclic CM hypergraph with edges $E_1, \ldots, E_n$. Let M be the set of given MVDs of H. $M \equiv \bowtie(E_1, \ldots, E_n)$. □

Example 8 In Figure 1, the given FDs are: $B \rightarrow M$, $M \rightarrow B$, $SC \rightarrow G$, and $C \rightarrow P$. The given MVDs are: $B \twoheadrightarrow A \mid M \mid SCGP$, $S \twoheadrightarrow BMA \mid CGP$, and $C \twoheadrightarrow P \mid SGBMA$. The given JD is: $\bowtie(AB, BM, SB, SCG, CP)$, which is equivalent to the given MVDs. Also, by Lemma 6, $S \twoheadrightarrow CG \mid B$ for the subset of vertices $SCGB$. ($SCGB$ is the set of attributes in the first scheme tree T in Figure 4 and is of particular interest when we apply the hypergraph-generated MVD $S \twoheadrightarrow BMA \mid CGP$ to T.) □

As a first important consequence of being able to apply standard dependency theory, we show that each edge in an acyclic CM hypergraph is in BCNF—a condition we need in order to guarantee that our algorithm generates redundancy-free scheme trees.

Lemma 8 Each edge of an acyclic CM hypergraph is in BCNF. □

2.3 Scheme Trees

Scheme trees are generic hierarchical structures. A scheme tree defines a nesting of a set of attributes, which, in turn, organizes the data values as a populated scheme tree according to the nesting.

Definition 9 A *scheme tree* T over a set U of attributes is a rooted tree in which every node is a non-empty subset of U. Further, the intersection of every pair of distinct nodes in T is empty. We denote the set of attributes in T by $Aset(T)$. □

Example 9 Figure 4 shows three scheme trees, T_1, T_2, and T_3, from left to right. Each of the nodes is non-empty and the intersection of every pair of distinct nodes within a single tree is empty. Intersections of nodes in different scheme trees, however, need not be empty: the vertex *Course*, for example, is in the left child node of T_1 and in the root node of T_2. As a textual representation of scheme trees, we use notation in which each subtree of a scheme tree is a repeating group of attribute names. Thus, the three scheme trees in textual notation are $T_1 = S(CG)^*(B)^*$, $T_2 = PC$, and $T_3 = BM(A)^*$, and thus $Aset(T_1) = SCGB$, $Aset(T_2) = PC$, and $Aset(T_3) = BMA$. □

Definition 10 Let T be a scheme tree over a set U of attributes. Let $dom(A)$ be the set of domain values of an attribute A in U and let $dom(U) = \bigcup_{A \in U} dom(A)$. A *populated scheme tree* over T is recursively defined as follows:

1. If T has only the root node $A_1 \ldots A_n$ ($n \geq 1$), a *populated scheme tree* over T is a (possibly empty) set of partial functions $\{t_1, \ldots, t_m\}$ from $\{A_1, \ldots, A_n\}$ to $dom(U)$ such that for each t_i, $t_i(A_j) \in dom(A_j)$ if t_i is defined on A_j.

2. If T has more than one node, then let $T_1, \ldots, T_n$ ($n \geq 1$) be the n subtrees of T such that the root node of each T_i is a child node of T's root node. If $\{t_1, \ldots, t_m\}$, $m \geq 0$, is the set of partial functions of T's root node, then r is a *populated scheme tree* over T if

   (a) $r = \bigcup_{j=1}^{m}(t_j \cup (\bigcup_{i=1}^{n} s_{j_i}))$ where s_{j_i} is the (possibly empty) populated scheme tree over T_i for the jth partial function t_j, and

   (b) for any non-empty s_{j_i}, there is a partial function t_j such that s_{j_i} is for t_j and t_j is defined on some attribute in T's root node, and

   (c) $t_p = t_q$ ($1 \leq p, q \leq m$) implies $s_{p_i} = s_{q_i}$ ($1 \leq i \leq n$). ($t_p = t_q$ if t_p and t_q are defined on exactly the same attributes, and $t_p(A) = t_q(A)$ if t_p and t_q are both defined on attribute A.) □

Example 10 As Figures 2–4 show, we can represent a populated scheme tree by embedded bucket notation in which each subtree in a scheme tree is a bucket nested in its parent bucket (with the outermost bucket being implicit). Every scheme tree in Figures 2–4 is correctly populated with respect to the given data instance in Figure 1. If, however, we were to add an activity a_1 for student s_1 in Figure 2, the populated scheme tree would violate Condition 2b of Definition 10 since activity a_1 would have no parent club and mascot. Or, if we were to replace the instance s_4 with s_3, the populated scheme tree would violate Condition 2c of Definition 10 since the two s_3's would have different subtrees under (*Club Mascot* (*Activity*)*)*. □

A forest of scheme trees for a given CM hypergraph must be able to store any valid data instance for the hypergraph. Thus, we now define "collectively covers" to capture exactly what it means for a forest of scheme trees to be able to include all the data and only all the data in a valid populated instance of a CM hypergraph.

Definition 11 Let $H = (V, E)$ be a connected acyclic CM hypergraph, and let $T_1, \ldots, T_n$ ($n \geq 1$) be scheme trees such that $Aset(T_1) \cup \cdots \cup Aset(T_n) = V$. Let $r_1, \ldots, r_n$ be populated instances of $T_1, \ldots, T_n$ respectively, and let nulls for $r_1, \ldots, r_n$ be unique within and across all populated instances. Let $\tau_1, \ldots, \tau_n$ be the respective total unnestings of $r_1, \ldots, r_n$, and let $r = \tau_1 \otimes \tau_2 \otimes \cdots \otimes \tau_n$ where for $1 < i \leq n$, $(Aset(T_1) \cup \ldots \cup Aset(T_{i-1})) \cap Aset(T_i) \neq \emptyset$, $T_1, \ldots, T_n$ *collectively cover* H if for any valid population H_D of data for H, $r = u$ up to renaming of nulls where u is a universal relation derived from H_D for H. □

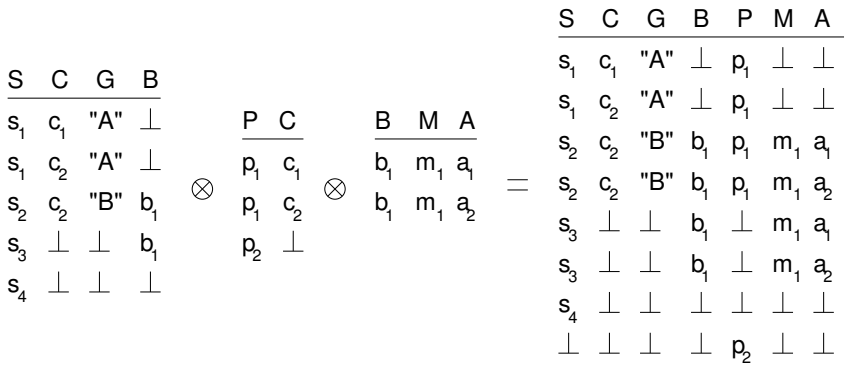

Figure 7: Outer join of the total unnestings of the populated scheme trees in Figure 4.

Example 11 The scheme trees in Figure 4 collectively cover the CM hypergraph and data instance in Figure 1, but the scheme tree in Figure 2 does not. Intuitively, it is clear that the scheme trees in Figure 4 include every value and every relationship in Figure 1. For Figure 2 this is not the case since p_2 does not appear anywhere. Formally, Figure 7 shows the outer join of the total unnestings of the populated scheme trees in Figure 4. Observe that, up to a renaming of nulls, the resulting relation is identical to the universal relation in Figure 6. □

2.4 Nested Normal Form

Nested Normal Form (NNF) [13] is a necessary and sufficient condition for removing data redundancy with respect to MVDs and FDs that hold for scheme trees. Since one of our goals is to generate redundancy-free scheme trees, they must all be in NNF. We now proceed to define NNF.[5]

Let T be a scheme tree. Let N be a node in T. Notationally, $Ancestor(N)$ denotes the union of attributes in all ancestors of N, including N. Similarly, $Descendant(N)$ denotes the union of attributes in all descendants of N, including N. Each edge (V, W) in T, where V is the parent of W, denotes an MVD $Ancestor(V) \twoheadrightarrow Descendant(W)$. Notationally, we use $MVD(T)$ to denote the set of all MVDs represented by the edges in T. By construction, each MVD in $MVD(T)$ is satisfied in the total unnesting of any populated instance of T.

Definition 12 Let U be a set of attributes. Let M be a set of MVDs over U and F be a set of FDs over U. Let T be a scheme tree such that $Aset(T) \subseteq U$. Let D_1 be the set of MVDs that hold for $Aset(T)$ with respect to $M \cup F$, and let D_2 be the set of FDs that hold for $Aset(T)$ with respect to $M \cup F$. T is in NNF with respect to $M \cup F$ if the following conditions are satisfied.

[5]NNF is based on standard dependency theory [10], which in turn is based on the URSA [14]. In general, CM hypergraphs do not satisfy the URSA and thus we could not make use of NNF directly in [11]. However, as Lemma 4 shows, the URSA holds in acyclic CM hypergraphs. Therefore, unlike in [11], we can use NNF directly in this paper.

1. $MVD(T) \cup D_2$ is equivalent to $D_1 \cup D_2$ on $Aset(T)$.

2. For each nontrivial FD $X \rightarrow A \in D_2$, $X \rightarrow Ancestor(N_A)$ also holds with respect to $M \cup F$, where N_A is the node in T that contains A. □

When NNF's Condition 1 is violated, there is a populated scheme tree that has redundancy with respect to an MVD that holds. When NNF's Condition 2 is violated, there is a populated scheme tree that has redundancy with respect to an FD that holds.

Example 12 All the scheme trees in Figures 3 and 4 are in NNF. As an example, let T be the scheme tree $S(CG)^*(B)^*$ in Figure 4. Thus, $Aset(T) = SCGB$ and $MVD(T) = \{S \twoheadrightarrow CG \mid B\}$. From Example 8, $S \twoheadrightarrow CGP \mid ABM$ are hypergraph-generated MVDs, which when restricted to $Aset(T)$ yields the set $D_1 = \{S \twoheadrightarrow CG \mid B\}$. (All other hypergraph-generated MVDs are trivial when restricted to $Aset(T)$.) $D_2 = \{SC \rightarrow G\}$ is the set of nontrivial FDs that hold on $Aset(T)$. Hence, since $MVD(T) \cup D_2 \equiv D_1 \cup D_2$ on $Aset(T)$, Condition 1 of NNF holds. NNF's Condition 2 clearly holds because for the given nontrivial FD $SC \rightarrow G$, $Ancestor(CG)$ is SCG and $SC \rightarrow SCG$. Similarly, we can argue that all other scheme trees in Figures 3 and 4 are in NNF.

On the other hand, the scheme tree in Figure 2 is not in NNF. To see the violations, let T be the scheme tree $S(BM(A)^*)^*(CGP)^*$ in Figure 2. Thus, $Aset(T) = SBMACGP$ and $MVD(T) = \{S \twoheadrightarrow BMA \mid CGP,\ SBM \twoheadrightarrow A\}$. The nontrivial MVDs and FDs that hold for $Aset(T)$ are $\{B \twoheadrightarrow A \mid M \mid SCGP,\ S \twoheadrightarrow BMA \mid CGP,\ C \twoheadrightarrow P \mid SGBMA\} \cup \{B \rightarrow M,\ M \rightarrow B,\ SC \rightarrow G,\ C \rightarrow P\}$. Because $B \twoheadrightarrow A$ does not follow from $MVD(T)$ and the FDs that hold for T, NNF's Condition 1 is violated. Consequently, there can be a populated instance of T that has redundant data with respect to $B \twoheadrightarrow A$. The populated scheme tree in Figure 2 is an example that demonstrates the redundancy caused by $B \twoheadrightarrow A$. In addition, $C \rightarrow P$ is an FD that holds for T but $C \not\rightarrow SCGP$. As a result, NNF's Condition 2 is also violated, and therefore there can be a populated instance of T that has redundant data with respect to $C \rightarrow P$. The populated scheme tree in Figure 2 also demonstrates the redundancy caused by $C \rightarrow P$. □

3 Scheme Tree Generation Algorithms

Given a connected acyclic CM hypergraph H, we begin by forming equivalence classes of edges based on functional equivalence of FDs given in H. We then observe that the set of vertices in each of the equivalence classes is in 4NF so that they are all redundancy-free with respect to the FDs and MVDs of H. Our goal is to combine the equivalence classes of H together into as few scheme trees as possible while retaining the redundancy-free property of NNF and while ensuring that the generated forest of scheme trees collectively covers H.

Algorithm 1 in this section generates the equivalence classes of H. Algorithm 2 organizes these equivalence-classes into a partial ordering that indicates which equivalence classes can potentially be combined in parent-child relationships to form larger redundancy-free scheme trees. Based on the partial ordering, shared vertices in disjoint components of the partial ordering, and the mandatory/optional constraints of H, Algorithm 3 combines equivalence classes of H as nodes into a minimal forest F of redundancy-free scheme trees that collectively cover H. Needing to know which equivalence classes can actually be child nodes of which other equivalence classes and not violate the collectively-covers property, Algorithm 3 repeatedly calls on Algorithm 4 to assist in its decision. So as to ensure that the algorithms run in polynomial time, Algorithms 3 and 4 are greedy in their generation of F.

3.1 Algorithm 1: Equivalence Classes

Let H be a connected acyclic CM hypergraph, and let E_i $(1 \leq i \leq n)$ be the n edges of H, each of which is a set of vertices. Let $\equiv_E$ be the relation over the edges of H such that $E_i \equiv_E E_j$ if $E_i \rightarrow E_j$ and $E_j \rightarrow E_i$. Based on Armstrong's axioms, the relation $\equiv_E$ is clearly reflexive, symmetric, and transitive, and thus an equivalence relation. Algorithm 1 builds the equivalence classes of a connected acyclic CM hypergraph H in a straightforward way. $E_i \rightarrow E_j$ and $E_j \rightarrow E_i$ if and only if the closure of the attribute sets of E_i and E_j are equal (i.e., if and only if $E_i^+ = E_j^+$).

Algorithm 1 Build equivalence classes of edges.

input: a connected acyclic CM hypergraph H
output: a set EqC of $\equiv_E$ equivalence classes of edges in H
1: for each edge E_i, compute the functional closure E_i^+
2: let EqC be a partition P of the edges of H such that edges E_i and E_j are in the same block of P if $E_i^+ = E_j^+$

Example 13 As a running example of our algorithmic generation of redundancy-free minimal scheme-tree forests in polynomial time, we introduce the CM hypergraph in Figure 8. Table 1 lists the CM-hypergraph's equivalence classes along with their edges and vertices. □

We let the notation $\overline{[E_i]}$ denote the set of vertices in the edges of the equivalence class $[E_i]$. Lemma 9 makes the important observation that each vertex set, $\overline{[E_i]}$, constitutes the largest possible 4NF relation scheme for flat relations, and thus also the largest possible set of attributes for individual nodes of scheme trees.

Lemma 9 Let H be a connected acyclic CM hypergraph. Let $[E_i]$ be an equivalence class of H. $\overline{[E_i]}$ is a largest possible set of vertices that does not violate 4NF. □

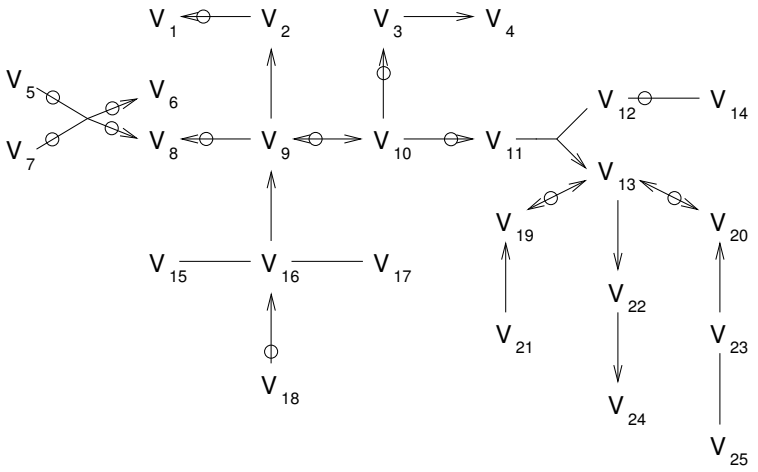

Figure 8: A sample connected acyclic CM hypergraph.

E_i	$[E_i]$	$\overline{[E_i]}$
$V_5V_7 \rightarrow V_6V_8$	$\{V_5V_7 \rightarrow V_6V_8\}$	$V_5V_6V_7V_8$
$V_2 \rightarrow V_1$	$\{V_2 \rightarrow V_1\}$	V_1V_2
$V_3 \rightarrow V_4$	$\{V_3 \rightarrow V_4\}$	V_3V_4
$V_{12}V_{14}$	$\{V_{12}V_{14}\}$	$V_{12}V_{14}$
$V_{11}V_{12} \rightarrow V_{13}$	$\{V_{11}V_{12} \rightarrow V_{13}\}$	$V_{11}V_{12}V_{13}$
$V_9 \leftrightarrow V_{10}$	$\{V_9 \rightarrow V_8, V_9 \rightarrow V_2, V_9 \leftrightarrow V_{10}, V_{10} \rightarrow V_3, V_{10} \rightarrow V_{11}\}$	$V_2V_3V_8V_9V_{10}V_{11}$
$V_{16} \rightarrow V_9$	$\{V_{16} \rightarrow V_9\}$	V_9V_{16}
$V_{15}V_{16}$	$\{V_{15}V_{16}\}$	$V_{15}V_{16}$
$V_{16}V_{17}$	$\{V_{16}V_{17}\}$	$V_{16}V_{17}$
$V_{18} \rightarrow V_{16}$	$\{V_{18} \rightarrow V_{16}\}$	$V_{16}V_{18}$
$V_{19} \leftrightarrow V_{13}$	$\{V_{19} \leftrightarrow V_{13}, V_{13} \leftrightarrow V_{20}, V_{13} \rightarrow V_{22}\}$	$V_{13}V_{19}V_{20}V_{22}$
$V_{21} \rightarrow V_{19}$	$\{V_{21} \rightarrow V_{19}\}$	$V_{19}V_{21}$
$V_{22} \rightarrow V_{24}$	$\{V_{22} \rightarrow V_{24}\}$	$V_{22}V_{24}$
$V_{23} \rightarrow V_{20}$	$\{V_{23} \rightarrow V_{20}\}$	$V_{20}V_{23}$
$V_{23}V_{25}$	$\{V_{23}V_{25}\}$	$V_{23}V_{25}$

Table 1: The $\equiv_E$ equivalence classes of the CM hypergraph in Figure 8.

3.2 Algorithm 2: Partial Ordering

In this section we begin showing how to combine equivalence classes by producing a partial ordering over the $\equiv_E$ equivalence classes. We then show that the Hasse diagram for the partial ordering yields the potential parent-child relationships of $\equiv_E$ equivalence classes in non-degenerate scheme trees.

Let H be a connected acyclic CM hypergraph, and let $[E_i]$ $(1 \leq i \leq n)$ be the n $\equiv_E$ equivalence classes of H. Let $\prec_{EqC}$ be the relation $[E_i] \prec_{EqC} [E_j]$ if $\overline{[E_i]} \rightarrow \overline{[E_j]}$. Then, $\prec_{EqC}$ induces a partial ordering over the equivalence classes of H. By Armstrong's axioms, the relation $\prec_{EqC}$ is clearly reflexive and transitive. It is also antisymmetric for if not then there are distinct equivalence classes $[E_i]$ and $[E_j]$ such that $\overline{[E_i]} \rightarrow \overline{[E_j]}$ and $\overline{[E_j]} \rightarrow \overline{[E_i]}$; but then there must be edges $E_p \in [E_i]$ and $E_q \in [E_j]$ such that $E_p \rightarrow E_q$ and $E_q \rightarrow E_p$—

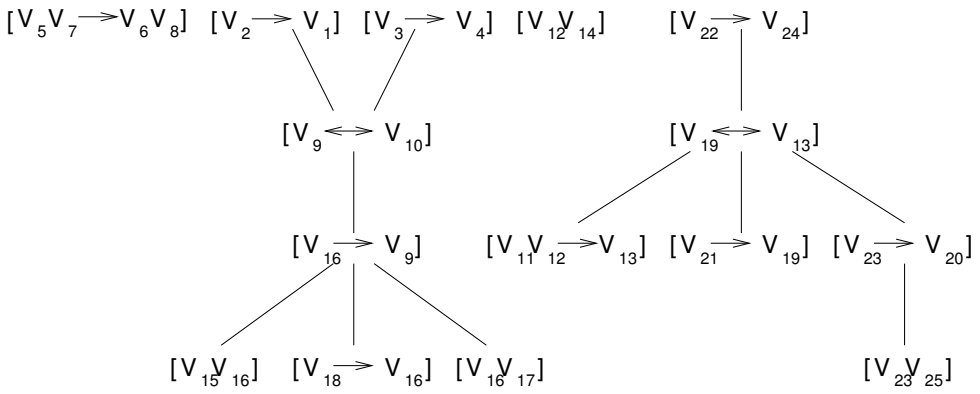

Figure 9: The Hasse diagram for the $\prec_{EqC}$ partial ordering.

a contradiction, since then $[E_i]$ and $[E_j]$ are not distinct. Algorithm 2 generates a Hasse diagram for the partial ordering over the set of $\equiv_E$ equivalence classes EqC produced by Algorithm 1 in a straightforward way. If the vertex sets of any two equivalence classes C_i and C_j intersect, then C_i and C_j are unrelated if there is no FD between $\overline{C_i}$ and $\overline{C_j}$ or, if there is an FD between them, the equivalence class of the right-hand side of the FD is the direct parent of the equivalence class of the left-hand side.

Algorithm 2 Build Hasse diagram for the equivalence classes of edges.

input: a set EqC of $\equiv_E$ equivalence classes of edges in H
output: a Hasse diagram D for the partial ordering $\prec_{EqC}$

```
1: let D initially be EqC
2: for i = 1 to |D| do
3:     for j = i + 1 to |D| do
4:         if C̄_i ∩ C̄_j ≠ ∅ then
5:             if C̄_i → C̄_j then
6:                 make C_j an immediate parent of C_i
7:             else if C̄_j → C̄_i then
8:                 make C_i an immediate parent of C_j
```

Example 14 Figure 9 shows the Hasse diagram for the partial ordering induced over the equivalence classes in Table 1, which are derived from Figure 8. Note that the partial ordering has five maximal equivalence classes: $[V_5V_7 \rightarrow V_6V_8]$, $[V_2 \rightarrow V_1]$, $[V_3 \rightarrow V_4]$, $[V_{12}V_{14}]$, and $[V_{22} \rightarrow V_{24}]$. □

An $\prec_{EqC}$ Hasse diagram has some interesting properties, which serve us well in building scheme trees. First, a connected acyclic CM hypergraph has a unique $\prec_{EqC}$ Hasse diagram. Second, each node in an $\prec_{EqC}$ Hasse diagram has a key, and parent-child relationships in a Hasse diagram connect through keys. These properties let us conclude that any parent-child relationship can form a redundancy-free parent-child connection in a scheme tree. The definitions and lemmas in the remainder of this subsection establish these properties of $\prec_{EqC}$ Hasse diagrams.

Lemma 10 The Hasse diagram D produced by Algorithms 1 and 2 is unique for a connected acyclic CM hypergraph H. □

Definition 13 Let H be a connected acyclic CM hypergraph. Let $[E_i]$ be an $\equiv_E$ equivalence class of edges of H. A set of vertices $K \subseteq \overline{[E_i]}$ is a *key* of $[E_i]$ if $K \rightarrow \overline{[E_i]}$. A key K is *minimal* if no proper subset of K is a key. □

Definition 14 Let H be a connected acyclic CM hypergraph. Let $[E_i]$ be an $\equiv_E$ equivalence class of edges of H. A vertex V of an edge in $[E_i]$ is a *key vertex* of $[E_i]$ if V is a key of $[E_i]$. □

Definition 15 Let H be a connected acyclic CM hypergraph. Let $[E_i]$ be an $\equiv_E$ equivalence class of edges of H. A vertex V of an edge in $[E_i]$ is a *connecting vertex* of $[E_i]$ if there is another equivalence class $[E_j]$ distinct from $[E_i]$ such that $\overline{[E_i]} \cap \overline{[E_j]} = V$. □

Example 15 Table 2 lists the minimal keys, and the key, non-key, and connecting vertices of the equivalence classes of Table 1. Note that although $V_5V_7 \rightarrow V_6V_8$ is a directed edge, its left-hand side is not a single vertex. Therefore, $[V_5V_7 \rightarrow V_6V_8]$ does not have a key vertex. The same is true for $[V_{11}V_{12} \rightarrow V_{13}]$. A non-directed edge, such as $V_{12}V_{14}$, cannot have a key vertex, since its minimal key has more than one vertex. □

$[E_i]$	minimal keys	key vertices	non-key vertices	connecting vertices
$[V_5V_7 \rightarrow V_6V_8]$	V_5V_7		V_5, V_6, V_7, V_8	V_8
$[V_2 \rightarrow V_1]$	V_2	V_2	V_1	V_2
$[V_3 \rightarrow V_4]$	V_3	V_3	V_4	V_3
$[V_{12}V_{14}]$	$V_{12}V_{14}$		V_{12}, V_{14}	V_{12}
$[V_{11}V_{12} \rightarrow V_{13}]$	$V_{11}V_{12}$		V_{11}, V_{12}, V_{13}	V_{11}, V_{12}, V_{13}
$[V_9 \leftrightarrow V_{10}]$	V_9, V_{10}	V_9, V_{10}	V_2, V_3, V_8, V_{11}	V_2, V_3, V_8, V_9, V_{11}
$[V_{16} \rightarrow V_9]$	V_{16}	V_{16}	V_9	V_9, V_{16}
$[V_{15}V_{16}]$	$V_{15}V_{16}$		V_{15}, V_{16}	V_{16}
$[V_{16}V_{17}]$	$V_{16}V_{17}$		V_{16}, V_{17}	V_{16}
$[V_{18} \rightarrow V_{16}]$	V_{18}	V_{18}	V_{16}	V_{16}
$[V_{22} \rightarrow V_{24}]$	V_{22}	V_{22}	V_{24}	V_{22}
$[V_{19} \leftrightarrow V_{13}]$	V_{13}, V_{19}, V_{20}	V_{13}, V_{19}, V_{20}	V_{22}	V_{13}, V_{19}, V_{20}, V_{22}
$[V_{21} \rightarrow V_{19}]$	V_{21}	V_{21}	V_{19}	V_{19}
$[V_{23} \rightarrow V_{20}]$	V_{23}	V_{23}	V_{20}	V_{20}, V_{23}
$[V_{23}V_{25}]$	$V_{23}V_{25}$		V_{23}, V_{25}	V_{23}

Table 2: Key, non-key, and connecting vertices of the equivalence classes of Table 1.

In Lemma 11, we observe something significant about the connection between parent and child nodes in Hasse diagrams over $\equiv_E$ equivalences classes. They connect through a single vertex that is a key vertex in the parent and a non-key vertex in the child.

Lemma 11 Let H be a connected acyclic CM hypergraph, and let D be the Hasse diagram for the partial ordering $\prec_{EqC}$ of $\equiv_E$ equivalence classes of H. $[E_p]$ is a parent of child $[E_c]$ in D if and only if $\overline{[E_p]} \cap \overline{[E_c]}$ is a single vertex V, and V is a key vertex of $[E_p]$ and a non-key vertex of $[E_c]$. □

Example 16 Consider the CM hypergraph H in Figure 8, the Hasse diagram D in Figure 9, and the connecting vertices between parent and child in Table 2. The single connecting vertex between the parent $[V_2 \rightarrow V_1]$ and child $[V_9 \leftrightarrow V_{10}]$ in D is V_2, and V_2 is a key for the parent $[V_2 \rightarrow V_1]$, but not for the child $[V_9 \leftrightarrow V_{10}]$. Similarly, V_3, V_9, and V_{16} are all single connecting vertices between parent and child and are also keys in their respective parent equivalence classes, but not in their respective child equivalence classes. □

Lemma 12 Let H be a connected acyclic CM hypergraph, and let D be the Hasse diagram for the partial ordering $\prec_{EqC}$ of $\equiv_E$ equivalence classes of H. For any FD $X \rightarrow A$ derived from the directed edges of H that applies to a vertex set $\overline{[E_i]}$ of an $\equiv_E$ equivalence class $[E_i]$ in D, $X \rightarrow \mathit{Ancestor}(\overline{[E_i]})$, where, here, $\mathit{Ancestor}(\overline{[E_i]})$ denotes the union of $\overline{[E_i]}$ with the sets of vertices of all upper-bound nodes of $[E_i]$ up to and including all maximal upper-bound nodes of $[E_i]$. □

Example 17 In Figure 9, $V_9 \rightarrow V_8$ is a given FD that applies to the vertex set of the equivalence class $[V_9 \leftrightarrow V_{10}]$. $\mathit{Ancestor}(\overline{[V_9 \leftrightarrow V_{10}]})$ is $\overline{[V_9 \leftrightarrow V_{10}]} \cup \overline{[V_2 \rightarrow V_1]} \cup \overline{[V_3 \rightarrow V_4]}$ and since $V_9 \rightarrow \overline{[V_9 \leftrightarrow V_{10}]}$, $V_9 \rightarrow \overline{[V_2 \rightarrow V_1]}$, and $V_9 \rightarrow \overline{[V_3 \rightarrow V_4]}$ all hold, $V_9 \rightarrow \mathit{Ancestor}(\overline{[V_9 \leftrightarrow V_{10}]})$ holds. The FD $V_{16} \rightarrow V_9$ is a given FD in node $[V_{16} \rightarrow V_9]$ and $V_{16} \rightarrow \mathit{Ancestor}(\overline{[V_{16} \rightarrow V_9]})$ holds; and for $V_{18} \rightarrow V_{16}$, $V_{18} \rightarrow \mathit{Ancestor}(\overline{[V_{18} \rightarrow V_{16}]}$ holds. For the node $[V_5V_7 \rightarrow V_6V_8]$, $V_5V_7 \rightarrow V_6$ is an FD derived from the given FD $V_5V_7 \rightarrow V_6V_8$, $\mathit{Ancestor}(\overline{[V_5V_7 \rightarrow V_6V_8]})$ is $\overline{[V_5V_7 \rightarrow V_6V_8]}$, and $V_5V_7 \rightarrow \mathit{Ancestor}(\overline{[V_5V_7 \rightarrow V_6V_8]})$ holds. □

Lemma 13 Let H be a connected acyclic CM hypergraph, and let D be the Hasse diagram for the partial ordering $\prec_{EqC}$ of $\equiv_E$ equivalence classes of H. Let $[E]$ be an equivalence class in D, and let A be a key vertex of $[E]$. Let Y be the union of $\overline{[E]}$ with the sets of vertices of all upper-bound nodes of $[E]$ excluding A. Let $[E']$ be a child equivalence class of $[E]$ in D, and let Z be the union of $\overline{[E']}$ with the sets of vertices of all lower-bound nodes of $[E']$. Finally, let S be AYZ. The FD $A \rightarrow Y$ and the MVD $A \twoheadrightarrow Z$ hold for S with respect to the hypergraph-generated MVDs of H. □

Example 18 In Figure 9, consider the child equivalence class $[V_9 \leftrightarrow V_{10}]$ and its parent $[V_2 \rightarrow V_1]$. Then, A in Lemma 13 is V_2, Y is V_1, Z is $V_3V_8V_9V_{10}V_{11}V_{15}V_{16}V_{17}V_{18}$, and S is AYZ. $V_2 \rightarrow V_1$ is a given FD and thus $A \rightarrow Y$ clearly holds. In Figure 8, the removal of V_2 ($= A$) yields the hypergraph-generated MVD $V_2 \twoheadrightarrow V_3 \ldots V_{25}$. Intersecting $V_3 \ldots V_{25}$ with S yields $V_3V_8V_9V_{10}V_{11}V_{15}V_{16}V_{17}V_{18}$, and thus $A \twoheadrightarrow Z$ holds. □

Lemmas 12 and 13 indicate that when building scheme trees using parent-child links from an $\prec_{EqC}$ Hasse diagram, the scheme trees will be redundancy free. This follows because Lemma 13 indicates that Condition 1 of NNF (Definition 12) will be satisfied, and Lemma 12 indicates that Condition 2 will be satisfied. Although not necessarily trees, each connected component of an $\prec_{EqC}$ Hasse diagram exhibits properties that satisfy the definition of NNF. After giving Algorithms 3 and 4, which produce scheme trees based on $\prec_{EqC}$ Hasse diagrams, we use Lemmas 12 and 13 as part of the proof to enable us to conclude that generated scheme trees are in NNF and are thus redundancy free.

3.3 Algorithms 3 and 4: Scheme-Tree Generation

Algorithm 3 generates the fewest redundancy-free scheme trees that collectively cover a given connected acyclic CM hypergraph. Its major **while** loop (Lines 4–34) generates so-called *transitional trees*, and its final **for** loop (Lines 35–38) transforms each transitional tree into a scheme tree. In essence, the first part of the major **while** loop (Lines 5–17) establishes roots for transitional trees, while the second part (Lines 18–33) grows trees as large as possible. As guided by the $\prec_{EqC}$ Hasse diagram and the mandatory and optional constraints of the given CM hypergraph, the algorithm puts every $\equiv_E$ equivalence class as a node in some growing transitional tree. Following the parent-child relationships in the Hasse diagram ensures that the generated scheme trees are redundancy free, while maximally adding nodes to a growing tree depends on joinability and data-coverage properties as dictated by optional and mandatory constraints checked in Algorithm 4.

Example 19 As an example, consider applying Algorithms 3 and 4 to the CM hypergraph H in Figure 8. Initially, the set F of transitional trees is empty, and G is the set of maximal equivalences classes of the working copy of a Hasse diagram D in Figure 9.

To find roots of new transitional trees, Algorithm 3 looks for maximal equivalence classes in G that have no non-key connecting vertices or exactly one in the remaining edges of the working copy of H (initially, H itself). A cross-check of Table 2 and G reveals that $[V_2 \rightarrow V_1]$, $[V_3 \rightarrow V_4]$, and $[V_{22} \rightarrow V_{24}]$ are maximal equivalence classes that have no non-key connecting vertices. They thus, in Line 8, become roots of transitional trees (and they appear as roots in the eventual full set of transitional trees in Figure 10). Line 8 also marks every vertex in these roots *coveredHere*, meaning that all data values for these vertices in the populated CM hypergraph can be stored in the scheme tree. By Definition 10, root nodes of a populated scheme tree can always store all values for all vertices (even when they have no relationship to other values in vertices of the node such as professor p_2 in the second populated scheme tree in Figure 4).

The maximal equivalence classes that have exactly one non-key connecting vertex, $[V_5V_7 \rightarrow$

Algorithm 3 Generate the fewest redundancy-free scheme trees that collectively cover a given connected acyclic CM hypergraph.

input: the input CM hypergraph to Algorithm 1 and the output Hasse diagram of Algorithm 2
output: the fewest redundancy-free scheme trees that collectively cover the input CM hypergraph

1: let H be a copy of the input CM hypergraph to Algorithm 1
2: let D be a copy of the output Hasse diagram of Algorithm 2
3: let F initially be an empty set of transitional trees
4: **while** D is not empty **do**
5: let G be the set of current maximal equivalence classes in D
6: **for** each $[E_c] \in G$ **do**
7: **if** $[E_c]$ has no non-key connecting vertex with respect to the remaining edges of H **then**
8: add $[E_c]$ to F and mark every vertex of $[E_c]$ "coveredHere" in $[E_c]$
9: remove $[E_c]$ from D
10: **if** $[E_c]$ has exactly one non-key connecting vertex V with respect to the remaining edges of H **then**
11: call Algorithm 4 with $[E_c]$ and V (or proceed with a previous result for $[E_c]$ and V)
12: **if** the answer is "yes" **then**
13: add V to F and mark V "coveredHere"
14: **if** the answer is "no" **then**
15: add $[E_c]$ to F and mark every vertex of $[E_c]$ "coveredHere" in $[E_c]$
16: remove $[E_c]$ from D
17: remove every edge in $[E_c]$ from H for each $[E_c] \in G$ that has been removed from D
18: **while** some remaining maximal equivalence classes in D are not marked **do**
19: let $[E_c]$ be an unmarked maximal equivalence class in D and mark it
20: **if** (1) $[E_c]$ has a parent $[E_p]$ that is a node in a tree T of F where V is their connecting vertex, or (2) $[E_c]$ has a non-key connecting vertex V that is a root node of a tree T of F **then**
21: call Algorithm 4 with $[E_c]$ and V (or proceed with a previous result for $[E_c]$ and V)
22: **if** the answer is "yes" and V is marked "coveredHere" **then**
23: mark each vertex of the returned set *coveredInThisEqClass* "coveredHere" in $[E_c]$
24: **if** Condition (1) above holds **then**
25: make $[E_c]$ a child node of $[E_p]$ in T
26: **else**
27: make $[E_c]$ a child node of V in T
28: remove $[E_c]$ from D (which may expose new unmarked maximal equivalence classes)
29: remove every edge in $[E_c]$ from H
30: **if** the answer is "yes" and V is *not* marked "coveredHere" **then**
31: remove the subtree rooted at $[E_p]$ and make it a new transitional tree in F (whose root node is $[E_p]$)
32: mark every vertex of $[E_p]$ "coveredHere" in $[E_p]$
33: remove from F any single-vertex degenerate transitional tree (caused by Line 31)
34: unmark every marked maximal equivalence class in D
35: **for** each transitional tree $T \in F$ **do**
36: replace each equivalence class $[E_i]$ in T by $\overline{[E_i]}$
37: **for** each child node $\overline{[E_c]}$ in T **do**
38: replace $\overline{[E_c]}$ by $\overline{[E_c]} - V$ where V is $[E_c]$'s non-key connecting vertex to its parent in T

$V_6V_8]$ and $[V_{12}V_{14}]$, are passed in Line 11 to Algorithm 4 along with their non-key connecting vertices. The purpose of the check in Algorithm 4 is to ensure that the generated set of scheme trees will collectively cover all the data values in any valid populated CM hypergraph. Algorithm 4 determines whether a node can be attached as a child: it can if all the values can be *coveredInThisEquivalenceClass* (i.e., covered when the node is a child node in a transitional tree) or can be *coveredElsewhere* (i.e., covered when the constraints of the CM hypergraph ensure that values that may not be covered in the child node itself will assuredly be covered in some other node). Algorithm 4 returns "no" for $[V_5V_7 \rightarrow V_6V_8]$ and V_8—"no" because at least one of the vertices, V_5, V_6, and V_7, is optional (indeed, all three are optional) and thus not *coveredInThisEqClass* nor *coveredElsewhere*. As a result, since the node cannot be a child node and at the same time maintain the collectively covers property,

Algorithm 4 Determine if an equivalence class $[E_c]$ can become a child node of a node that contains one of $[E_c]$'s non-key connecting vertices.

input: an equivalence class $[E_c]$ and a non-key connecting vertex V of $[E_c]$
output: "yes" or "no"

```
 1: let C be a copy of [E_c]
 2: let coveredInThisEqClass initially be an empty set of vertices
 3: let coveredElsewhere initially be an empty set of vertices
 4: in C designate the input connecting vertex V "joinable"
 5: while C has an unmarked edge E_i that contains a "joinable" vertex V_j do
 6:     mark E_i in C
 7:     for each vertex V_k ∈ E_i such that V_k ≠ V_j do
 8:         if the V_k connection in E_i is mandatory then
 9:             add V_k to coveredInThisEqClass
10:             designate V_k "joinable"
11:         else
12:             if in a transitional tree in F, V_k is a root node or a vertex of an equivalence-class root node then
13:                 add V_k to coveredElsewhere
14:             if V_k has a mandatory participation in an edge that is not in C then
15:                 add V_k to coveredElsewhere
16: if C has an unmarked edge then
17:     return "no"
18: if C has a vertex that is not the input connecting vertex V and not in (coveredInThisEqClass ∪ coveredElsewhere) then
19:     return "no"
20: return "yes" and the set coveredInThisEqClass
```

the algorithm makes $[V_5V_7 \rightarrow V_6V_8]$ a root node in Line 15, which ensures that every data value for all of its vertices is covered. On the other hand, Algorithm 4 returns "yes" for $[V_{12}V_{14}]$ and V_{12} because the connection to V_{14} is mandatory. Therefore, $[V_{12}V_{14}]$ can be a child node, but only a child node of the vertex V_{12} since $[V_{12}V_{14}]$ is a maximal equivalence class in the working copy of D. Thus, the statement in Line 13 adds V_{12} to F, which makes it a root of a transitional tree, and thus also *coveredHere*.

F is now the set of root nodes $\{[V_5V_7 \rightarrow V_6V_8]$, $[V_2 \rightarrow V_1]$, $[V_3 \rightarrow V_4]$, V_{12}, $[V_{22} \rightarrow V_{24}]\}$, the first four root nodes and the last root node in the eventual set of transitional trees in Figure 10. Further, in Lines 9 and 16, $[V_5V_7 \rightarrow V_6V_8]$ $[V_2 \rightarrow V_1]$, $[V_3 \rightarrow V_4]$, and $[V_{22} \rightarrow V_{24}]$ will have been removed from the working Hasse diagram D along with their edges, in Line 17, from the working CM hypergraph H. Hence, $[V_9 \leftrightarrow V_{10}]$ and $[V_{19} \leftrightarrow V_{13}]$ will have become the maximal equivalence classes along with $[V_{12}V_{14}]$, which remains, having not been removed even though one of its vertices has become a root node.

The **while** loop beginning at Line 18 adds child nodes to the root nodes in F so long as the nodes pass the check in Algorithm 4 and thus maintain the collectively covers property. Selecting $[V_{12}V_{14}]$ at Line 19 makes $[V_{12}V_{14}]$ a child node of V_{12} in Line 27 because Algorithm 4 has already been called for $[V_{12}V_{14}]$ and V_{12} and has returned "yes" and because V_{12} was marked *coveredHere* when it was added to F in Line 13. Similarly, albeit with Condition (1) rather than (2), selecting $[V_9 \leftrightarrow V_{10}]$ in the next iteration of the **while** loop at Line 18 makes $[V_9 \leftrightarrow V_{10}]$ a child of $[V_2 \rightarrow V_1]$. Note that $[V_9 \leftrightarrow V_{10}]$ cannot be a child of $[V_3 \rightarrow V_4]$ since V_9 values might not be covered because of the optional constraint on V_9 in Figure 8. If the optional constraint were mandatory instead, $[V_9 \leftrightarrow V_{10}]$ could be a child of either

$[V_3 \rightarrow V_4]$ or $[V_2 \rightarrow V_1]$, and either choice would maintain the minimality of the generated scheme-tree forest. Continuing to build on $[V_9 \leftrightarrow V_{10}]$, the algorithm adds $[V_{16} \rightarrow V_9]$ to the growing transitional tree and then subsequently $[V_{15}V_{16}]$ and $[V_{16}V_{17}]$. It does not, however, add $[V_{18} \rightarrow V_{16}]$ because the optional constraint on V_{18} can disallow valid instances of V_{18} from being stored and thus violate collectively covers.

At this point $[V_{19} \leftrightarrow V_{13}]$ is yet another maximal equivalence class in D to consider, so the **while** loop at Line 18 continues. For the vertex V_{22}, which appears in the equivalence class $[V_{19} \leftrightarrow V_{13}]$ and is a connecting vertex to the root node $[V_{22} \rightarrow V_{24}]$ in F, Algorithm 4 returns "yes." Thus, since V_{22} has been marked *coveredHere* in Line 8 and since it is Condition (1) that holds, $[V_{19} \leftrightarrow V_{13}]$ becomes a child node of $[V_{22} \rightarrow V_{24}]$—which is particularly interesting since, as will be seen, $[V_{19} \leftrightarrow V_{13}]$ does not end up as a child of $[V_{22} \rightarrow V_{24}]$ as Figure 10 shows. Nevertheless, for the current state in the algorithm, $[V_{19} \leftrightarrow V_{13}]$ is a child node of $[V_{22} \rightarrow V_{24}]$ and removal of $[V_{19} \leftrightarrow V_{13}]$ from D exposes three more maximal equivalence classes—$[V_{11}V_{12} \rightarrow V_{13}]$, $[V_{21} \rightarrow V_{19}]$, and $[V_{23} \rightarrow V_{20}]$. Selecting $[V_{11}V_{12} \rightarrow V_{13}]$ makes it a child of $[V_{19} \leftrightarrow V_{13}]$. However, it could also be a child of V_{12} in F and only is not because of the arbitrary choice of considering Condition (1) before Condition (2) when both hold as they do for $[V_{11}V_{12} \rightarrow V_{13}]$. Selecting either $[V_{21} \rightarrow V_{19}]$ or $[V_{23} \rightarrow V_{20}]$ next leads to an interesting situation. Neither can become a child of $[V_{19} \leftrightarrow V_{13}]$, but only because the *coveredHere* test for both V_{19} and V_{20} in $[V_{19} \leftrightarrow V_{13}]$ fails to hold. Hence, the **if** statement in Line 30 (rather than in Line 22) executes, which makes the subtree rooted at $[V_{19} \leftrightarrow V_{13}]$ a new transitional tree. Now since V_{19} and V_{20} are in a root node, their optional constraints no longer prohibit them from holding any and all values and the algorithm marks them *coveredHere*. This allows both $[V_{21} \rightarrow V_{19}]$ and $[V_{23} \rightarrow V_{20}]$ to become children of $[V_{19} \leftrightarrow V_{13}]$. Subsequently, $[V_{23}V_{25}]$ becomes a child of $[V_{23} \rightarrow V_{20}]$ resulting in the transitional tree rooted at $[V_{19} \leftrightarrow V_{13}]$ in Figure 10. Note that the alternative choice of *not* generating a new transitional tree rooted at $[V_{19} \leftrightarrow V_{13}]$ would make both $[V_{21} \rightarrow V_{19}]$ and $[V_{23} \rightarrow V_{20}]$ root nodes for two new transitional trees, which would unnecessarily increase the number of transitional trees in F (two additional instead of one additional).

Finally, the **while** loop at Line 18 terminates, and in Line 34 the algorithm unmarks $[V_{18} \rightarrow V_{16}]$, having considered it but having not been able to add it to any tree in the growing forest F of transitional trees. In the example, it now becomes the only remaining equivalence class for the next iteration of the **while** loop at Line 4. The set G of maximal equivalence classes at Line 5, thus, now consists of just the single remaining equivalence class $[V_{18} \rightarrow V_{16}]$. Since $[V_{18} \rightarrow V_{16}]$ is the only equivalence class left in D and its edges are the only ones left in H, $[V_{18} \rightarrow V_{16}]$ has no connecting vertex with respect to the remaining edges of the working copy of H. Hence, $[V_{18} \rightarrow V_{16}]$ becomes a root node in the final set of transitional trees F in Figure 10.

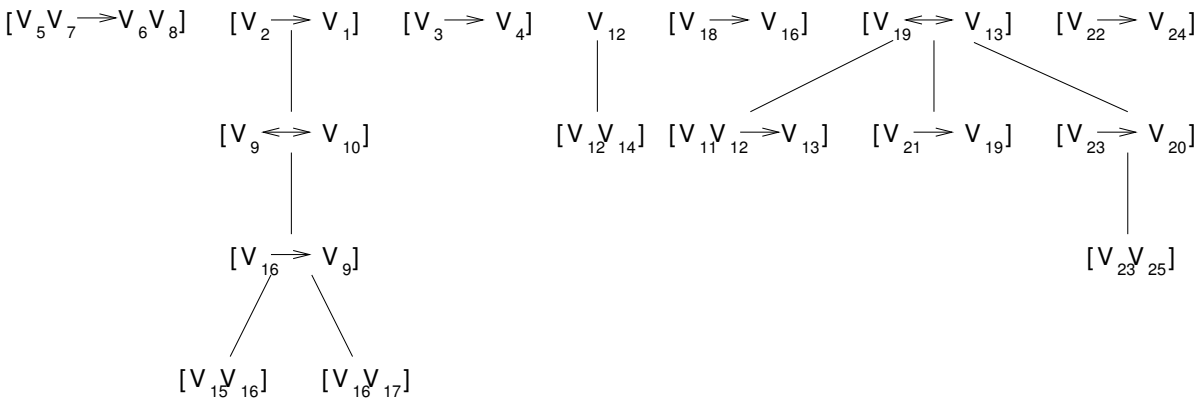

Figure 10: Generated transitional trees.

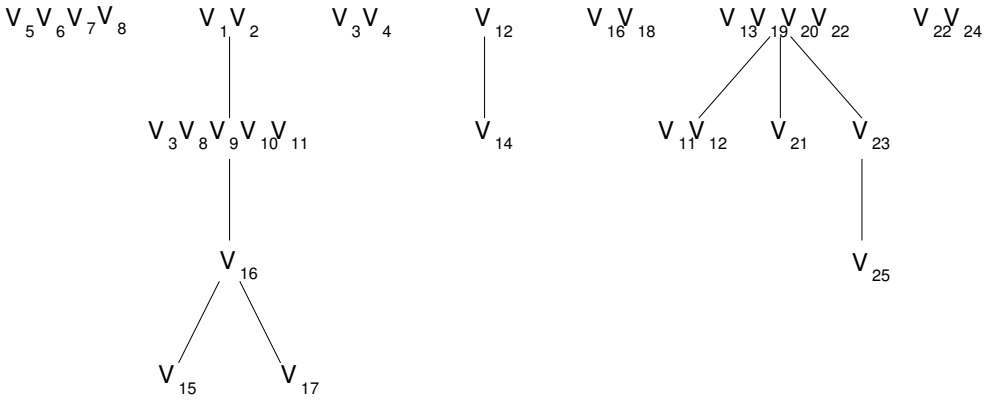

Figure 11: Generated scheme trees.

Lines 35–38 transform transitional trees into scheme trees, one for one. The statements simply replace each node in a transitional tree by the set of vertices in the transitional-tree node minus the vertex in each child node that connects it to its parent node. The result is the forest of scheme trees in Figure 11. □

4 Correctness and Tractability

For correctness, we must guarantee that Algorithms 1–4 generate scheme trees with two properties: (1) They disallow redundancy when storing any valid data instance of a given connected acyclic CM hypergraph (Section 4.1). (2) They collectively cover the data of the given hypergraph with as few scheme trees as possible (Section 4.2). For tractability, we must show that Algorithms 1–4 have polynomial-time complexity (Section 4.3).

4.1 Redundancy-Free Scheme Trees

In [13], we proved that a scheme tree is redundancy-free if and only if it is in NNF with respect to a given set of applicable FDs and MVDs. Hence, it suffices to show that Algorithms 1–4 generate NNF scheme trees with respect to the specified FD edges and the hypergraph-generated MVDs of a connected acyclic CM hypergraph.

Theorem 1 Let H be a connected acyclic CM hypergraph. Algorithms 1–4 generate NNF scheme trees from H.

Proof. (*sketch*) For a scheme tree T generated from H by Algorithms 1–4, let D_1 and D_2 respectively be the set of MVDs and the set of FDs that hold for $Aset(T)$ with respect to the hypergraph-generated MVDs and FDs of H. For Condition 1 of NNF, we must show the equivalence of $MVD(T) \cup D_2$ and $D_1 \cup D_2$: $D_1 \cup D_2$ implies $MVD(T)$ because each MVD in $MVD(T)$ is implied by an MVD in D_1 by augmentation. For an MVD $A \twoheadrightarrow Y \in D_1$ where $A \rightarrow Y \in D_2$, D_2 implies $A \twoheadrightarrow Y$ immediately. For an MVD $A \twoheadrightarrow Y \in D_1$ where $A \rightarrow Y \notin D_2$, A must be a key vertex of the node that contains A, and thus $MVD(T) \cup D_2$ implies $A \twoheadrightarrow Y$. For Condition 2 of NNF we must show that $X \rightarrow Ancestor(N_A)$ for any nontrivial FD $X \rightarrow A \in D_2$: When $X \rightarrow A$ applies to scheme tree T, X is completely contained within some node N of T. If Algorithm 3 generates N as the child of a non-key connecting vertex V, then V is the root of T, X is a key for N, and V is a vertex in the equivalence class of N. Thus, $X \rightarrow V$. Further, A must either be V or be in N; in either case, $X \rightarrow Ancestor(N_A)$ holds. Otherwise, Algorithm 3 generates N as the child of a node N' in T, where N' comes from an $\equiv_E$ equivalence class of H. In this case, the equivalence class for N' is the parent of the equivalence class for N in the Hasse diagram produced by Algorithm 2. Hence, $\overline{N} \rightarrow Ancestor(N)$. Thus, since $X \subseteq \overline{N}$ and is a key for N and since A is either in N or in an ancestor of N, $X \rightarrow Ancestor(N_A)$ holds. □

4.2 Minimal NNF Scheme-Tree Forests

Minimal NNF scheme-tree forests have the fewest possible number of scheme trees, but not fewer than enough to hold all the data in any valid population of a given CM hypergraph H. Thus, we first provide Lemma 14, which shows that Algorithms 1–4 generate scheme trees from H that collectively cover H. We then proceed with Theorem 2, which, along with Lemma 15 shows that the number of scheme trees generated is the fewest possible.

Lemma 14 Let H be a connected acyclic CM hypergraph. Algorithms 1–4 generate scheme trees $T_1, \ldots, T_n$ from H that collectively cover H. □

Given that the Hasse diagram D for a given connected acyclic CM hypergraph H is unique (Lemma 10), and further that the $\equiv_E$ equivalence class nodes of D are as large as

possible (Lemma 9), we seek for a way to maximally combine the nodes of D in order to reduce the total number of generated scheme trees. In Lemma 15 we first observe that there are only two ways to combine nodes without violating either NNF or collectively covers. Then, Theorem 2 shows that Algorithm 3 does indeed maximally combine the nodes in these two ways yielding the fewest number of scheme trees.

Lemma 15 Let D be the $\prec_{EqC}$ Hasse diagram for the $\equiv_E$ equivalence classes of a connected acyclic CM hypergraph H. Without violating NNF and while satisfying collectively covers, we can combine nodes N_1 and N_2 in D in a transitional tree T with the following two combining operations and only with these two combining operations: (1) if N_1 is the parent of N_2 in D, N_1 can be the parent of N_2 in T if for all valid populations of H, the edges of N_2 join completely with N_1 and every vertex of N_2 has mandatory participation in an edge of H or is in the root node of a transitional tree, and (2) if N_1 and N_2 have a connecting vertex V such that V is a key of neither N_1 nor N_2, and if for all valid populations of H, the edges of both N_1 and N_2 join completely with V and every vertex of both N_1 and N_2 has mandatory participation in an edge of H or is in the root node of a transitional tree, then N_1 and N_2 can be combined in T by making V a root node and making both N_1 and N_2 children of V. □

Theorem 2 Let H be a connected acyclic CM hypergraph, and let D be the $\prec_{EqC}$ Hasse diagram for the $\equiv_E$ equivalence classes of H. Algorithm 3 maximally combines the nodes of D into a forest F of the fewest number of transitional trees whose corresponding scheme trees are in NNF and collectively cover H.

Proof. (*sketch*) The **if** statements in Lines 7, 10, and 30 determine when to initialize new trees needed to collectively cover H. These conditions guarantee that no unnecessary trees are initialized. Then, the **while** loop beginning at Line 18 maximally adds children to established trees in the only two ways possible (Lemma 15). It adds every node of the Hasse diagram that satisfies the conditions of Combining Operation (1) in Line 25, and it adds every node that satisfies the conditions of Combining Operation (2) in Line 27. The additions are maximal because further additions would violate either NNF or collectively covers. As a result, a minimum number of trees are generated. □

4.3 Polynomial-Time Complexity

Before showing that Algorithms 1–4 run in polynomial time, we first guarantee that they halt. That Algorithms 1, 2, and 4 halt is clear, but Algorithm 3 only halts if it removes all the equivalence classes in the Hasse Diagram. We can guarantee, however, that the outer loop of Algorithm 3 removes at least one equivalence class in each iteration. Although the

average case should be much better, we are able to fairly easily show an $O(n^3)$ bound on all the algorithms running together. In deriving this $O(n^3)$ bound, we make the simplifying assumption that the arity of the largest-degree edge of H is bounded by a constant b. In practice, b is usually small—typically three or four—and in any case constant for H.

Lemma 16 Algorithm 3 halts. □

Lemma 17 Let H be a connected acyclic CM hypergraph, and let H have n edges. Algorithm 1 has $O(n^2)$ complexity. □

Lemma 18 Let H be a connected acyclic CM hypergraph, and let H have n edges. Algorithm 2 has $O(n^2)$ complexity. □

Lemma 19 Let H be a connected acyclic CM hypergraph, and let H have n edges. Algorithm 4 has $O(n^2)$ complexity. □

Lemma 20 Let H be a connected acyclic CM hypergraph, and let H have n edges. Algorithm 3 has $O(n^3)$ complexity. □

Theorem 3 Let H be a connected acyclic CM hypergraph, and let H have n edges. Executing Algorithms 1–4 together has $O(n^3)$ complexity.

Proof. (*sketch*) Since each iteration the Line-4 **while** loop of Algorithm 3 removes at least one equivalence class and since inner loops consider at most all remaining maximal equivalence classes, the loops of Algorithm 3 have, at most, $O(m^2)$ complexity where m is the number of equivalence classes of H. Within the loops, however, since H is acyclic, there are at most $2(m-1)$ equivalences-class/vertex pairs to check in Algorithm 4. Thus, over the entire execution of a maximum of m iterations of the Line-4 while loop, since Algorithm 3 calls Algorithm 4 at most once for each equivalence-class/vertex pair, instead of a multiplicative combination when adding in the $O(n^2)$ of Algorithm 4, the Line-4 **while** loop is bounded by $O(mn^2)$. Since $m \leq n$, the complexity of Algorithm 3 is bounded by $O(n^3)$. □

Showing that Algorithms 1–4 run in polynomial time is sufficient for our objective. We point out that more efficient renditions of these algorithms, such as combining Algorithms 1 and 2, are possible. However, for ease of presentation and of understanding the salient points of our algorithms we avoid inter-tangling algorithms and adding data structures to more efficiently find and select among alternatives.

5 Discussion

Besides being theoretically interesting, Algorithms 1–4 also provide practical guidelines for design of XML storage structures: (1) create a conceptual-model (CM) hypergraph, (2)

make the hypergraph acyclic, (3) use Algorithms 1–4 to generate a good design, and (4) map scheme trees to XML schema specifications [7]. Designers, however, must use some care in creating the CM hypergraph and, possibly also, in guiding the scheme-generation process.

Creating the CM hypergraph. Designers must ensure that the input CM hypergraph is canonical. A CM hypergraph H is *canonical* if H is acyclic, has no redundant edges or edge components, and has no losslessly decomposable n-ary edges. Details about how to identify and remove redundant edges and edge components and about how to identify and decompose losslessly decomposable n-ary edges are in [6]. After removing redundant edges and edge components and losslessly decomposing n-ary edges, designers can break any remaining cycles by adding roles for one of the connections in a cycle. An alternative for cycles connecting two edges both with the same composite key (e.g., edge $E_1 = AB \rightarrow C$ and edge $E_2 = AB \rightarrow D$), is to replace the vertices in the composite key with a single vertex V, and let the composite key attributes be in a bijective relationship with V (e.g., $E_1 = V \rightarrow C$, $E_2 = V \rightarrow D$, and new edge $V \leftrightarrow AB$). One nice feature about the approach is that once a CM hypergraph is canonical, designers need not be concerned about either the universal relation-scheme assumption (URSA) or the universal relation assumption (URA) since they are guaranteed to hold (Lemmas 4 and 5 respectively).

Guiding the scheme-generation process. Although our algorithm yields a scheme-tree forest with the fewest, redundancy-free scheme trees, the generated scheme-tree forest is not necessarily unique. Some alternatives are mere rearrangements of one of the generated scheme trees. By default Algorithms 1–4 push attributes toward the root, but alternative non-redundant arrangements may be better. For example, although the algorithms generate the second scheme tree, PC, in Figure 4, a rearrangement instead as the first scheme tree, $P(C)$*, in Figure 3 may be preferable. Other alternatives allow for rearrangements yielding a different minimal set of scheme trees. For example, as mentioned in Example 19, the subtree rooted at $[V_9 \leftrightarrow V_{10}]$ in Figure 10 could instead have been a subtree of $[V_3 \rightarrow V_4]$ if the optional constraint on V_9 were mandatory. With a backtracking algorithm that considers all possibilities, it may be reasonable to run Algorithms 1–4 to generate all scheme-tree forests and then subjectively choose the best. Lastly, in some designs for XML storage structures, it is important that certain vertices be in root nodes. Fixing root nodes and then maximally combining nodes in accord with Lemma 15 can lead to good designs with constrained roots. We point out also that the scheme-generation algorithms allow for maximum flexibility by not requiring a standard primary key in root nodes that disallows nulls. If designers wish to ensure that root nodes have standard primary keys, they can adjust the optional/mandatory constraints in the input CM hypergraph.

6 Concluding Remarks

We have achieved our objective of showing how to generate the fewest redundancy-free scheme trees that collectively cover a given acyclic conceptual-model hypergraphs in polynomial time. Theorem 1 guarantees that Algorithms 1–4 generate NNF scheme trees, which are redundancy free; Theorem 2 guarantees that the number of scheme trees generated is minimal; and Theorem 3 guarantees that the algorithms run together in polynomial time. As a practical application, Algorithms 1–4 can provide a firm basis for a good design of desirable XML storage structures. Designers, however, still have ultimate control and can use the algorithms intelligently to arrive at the best possible designs.

Acknowledgements

The work described in this paper was partially supported by Strategic Research Grants 7001945 and 7002140 of City University of Hong Kong. D.W. Embley was supported in part by the National Science Foundation under grant numbers 0083127 and 0414644.

References

[1] Marcelo Arenas and Leonid Libkin. A normal form for XML documents. *ACM Transactions on Database Systems*, 29:195–232, 2004.

[2] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[3] Ronald Bourett. XML database products, March 2007. http://www.rpbourret.com/-xml/XMLAndDatabases.htm.

[4] Yi Chen, Susan Davidson, and Yifeng Zheng. RRXS: Redundancy reducing XML storage in relations. In *Proceedings of the 29th International Conference on Very Large Databases*, pages 189–200, Berlin, Germany, September 2003.

[5] Edgar F. Codd. Recent investigations in relational data base systems. In *Proceedings of IFIP Congress 74*, pages 1017–1021, Stockholm, Sweden, August 1974.

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

[7] David W. Embley and Win Yin Mok. Producing XML documents with guaranteed 'good' properties. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, volume IX, pages 195–198, Orlando, Florida, July 2003.

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

[9] Leonid Libkin. Normalization theory for XML. In *Proceedings of the 5th International XML Database Symposium*, pages 1–13, Vienna, Austria, September 2007.

[10] David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[11] Wai Yin Mok and David W. Embley. Generating compact redundancy-free XML documents from conceptual-model hypergraphs. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1082–1096, 2006.

[12] Wai Yin Mok, Joseph Fong, and David W. Embley. Extracting a largest redundancy-free XML storage structure from an acyclic hypergraph in polynomial time. *Information Systems*, 35(7):804–824, 2010.

[13] Wai Yin Mok, Yiu-Kai Ng, and David W. Embley. A normal form for precisely characterizing redundancy in nested relations. *ACM Transactions on Database Systems*, 21(1):77–106, 1996.

[14] Yehoshua Sagiv. A characterization of globally consistent databases and their correct access paths. *ACM Transactions on Database Systems*, 8(2):266–286, 1983.

[15] Klaus-Dieter Schewe. Redundancy, dependencies and normal forms for XML databases. In *Proceedings of the Sixteenth Australasian Database Conference*, pages 7–16, Newcastle, Australia, January/February 2005.

[16] Millist W. Vincent, Jixue Liu, and Chengfei Liu. Strong functional dependencies and their application to normal forms in XML. *ACM Transactions on Database Systems*, 29(3):445–462, 2004.

[17] Junhu Wang and Rodney W. Topor. Removing XML data redundancies using functional and equality-generating dependencies. In *Proceedings of the Sixteenth Australasian Database Conference*, pages 65–74, Newcastle, Australia, January/February 2005.

[18] Cong Yu and H. V. Jagadish. XML schema refinement through redundancy detection and normalization. *The VLDB Journal*, 17(2):203–223, 2008.