Extracting a Largest Redundancy-Free XML Storage Structure from an Acyclic Hypergraph in Polynomial Time

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

Abstract

Given a hypergraph and a set of embedded functional dependencies, we investigate the problem of determining the conditions under which we can efficiently generate redundancy-free XML storage structures with as few scheme trees as possible. Redundancy-free XML structures guarantee both economy in storage space and the absence of update anomalies, and having the least number of scheme trees requires the fewest number of joins to navigate among the data elements. We know that the general problem is intractable. The problem may still be intractable even when the hypergraph is acyclic and each hyperedge is in Boyce-Codd Normal Form (BCNF). As we show here, however, given an acyclic hypergraph with each hyperedge in BCNF, a polynomial-time algorithm exists that generates a largest possible redundancy-free XML storage structure. Successively generating largest possible scheme trees from among hyperedges not already included in generated scheme trees constitutes a reasonable heuristic for finding the fewest possible scheme trees. For many practical cases, this heuristic finds the set of redundancy-free XML storage structures with the fewest number of scheme trees. In addition to a correctness proof and a complexity analysis showing that the algorithm is polynomial, we also give experimental results over randomly generated but appropriately constrained hypergraphs showing empirically that the algorithm is indeed polynomial.

Keywords: XML data redundancy, large XML storage structures, XML-Schema generation, acyclic hypergraphs

1 Introduction

XML databases are emerging [5]. Two types of XML databases are native XML databases, whose backend storage structures are internal representations of XML documents, and XML-enabled databases, whose backend storage structures are internal representations of relational tables. The fundamental unit of (logical) storage in native XML databases is an XML document [4]. Thus, designing XML documents for efficient retrieval and update has been a topic of recent research

[*]Department of Economics and Information Systems, University of Alabama in Huntsville, Huntsville, Alabama 35899, USA, `mokw@email.uah.edu`. (Most of this research was conducted while W. Y. Mok was a Visiting Research Fellow at City University of Hong Kong.)

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

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

[9, 11, 12]. The fundamental unit of (logical) storage in XML-enabled databases is a relational table. This table-storage method requires various mapping rules to translate between XML document schemas and database schemas and employs middleware to transfer data between XML documents and databases [4, 20, 23]. A recent study shows that designing XML documents for efficient retrieval and update can also guarantee well-designed relational storage structures for XML-enabled databases [13]. Thus, for both native XML databases and XML-enabled databases, designing XML documents for efficient retrieval and update is an appropriate focus for study.

Similar to design of relational tables by normalizing relational schemas, designing XML documents for efficient retrieval and update is about normalizing XML storage schemas. Normalized XML storage schemas remove the possibility of redundancy with respect to constraints and typically make both retrieval and update more efficient. Thus, there has been a flurry of research work on normalization of XML documents [2, 6, 7, 15, 18, 22, 25, 26, 27].

This paper, which follows up on our previous work [7, 18], is another step in this direction. In [18] we showed that generating a minimum number of redundancy-free XML storage structures from a conceptual-model hypergraph is NP-hard. Here we consider special-case conditions[1] that commonly hold in practice in an effort to find an efficient algorithm. Since it is known that checking whether relational schemas are in Boyce-Code Normal Form (BCNF) is intractable, our first condition limits conceptual-model hypergraphs to those in which each hypergraph edge is in BCNF with respect to the given functional dependencies (FDs). Next, since cycles in hypergraphs introduce ambiguity and typically cause difficulties, we assume that conceptual-model hypergraphs are acyclic. Finally, we assume that the only multivalued dependencies (MVDs) are hypergraph-generated MVDs. Even with these assumptions, however, it is an open problem to find an algorithm that generates a minimum number of redundancy-free XML storage structures in polynomial time. We therefore settle on a heuristic that resolves the issue for many practical cases and likely gives good results for all cases.

As the basis of our heuristic, we provide in this paper a polynomial-time algorithm that generates a largest scheme tree from an acyclic hypergraph and a set of FDs where each FD is embedded in some hyperedge and each hyperedge is in BCNF. As an approximation to generating a minimum number of redundancy-free XML storage structures, we use this heuristic repeatedly on the remaining hypergraph edges not already included in generated scheme-tree storage structures. This heuristic always yields redundancy-free XML storage structures and often, especially in practical cases, yields the fewest.

To illustrate or our approach and to show some of the pitfalls involved, we present a motivating example. In this example, we rely on intuition for some undefined terms. Later in Section 2, we

[1] In making these special-case assumptions, we point out that many conceptual-model hypergraphs found in practice satisfy these assumptions without any need for modification. For those that do require some modification to satisfy these conditions, the modifications are often minimal and straightforward. (1) In practice, conceptual-model hypergraph edges rarely violate BCNF. Further, since the size of a edge is typically small, checking exhaustively for keys of the edge and for applicable non-trivial FDs is not inordinately expensive. (2) In practice, we can always introduce role attributes, as needed, to break cycles. (3) In practice, we almost never care about any MVDs except hypergraph-generated MVDs.

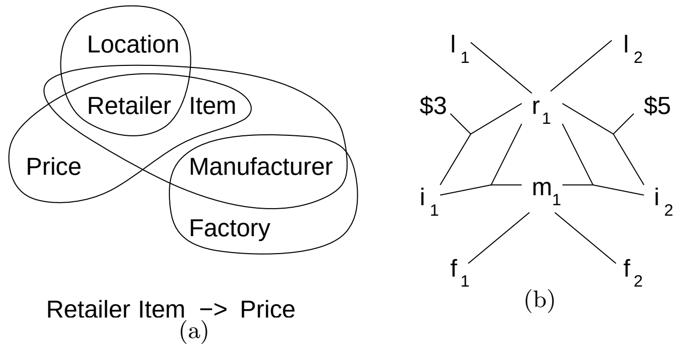


Figure 1: The Acyclic Hypergraph and Relationships of Example 1.

formally define these terms.

Example 1 Figure 1(a) shows an acyclic hypergraph and an FD, *Retailer Item* $\rightarrow$ *Price*, embedded in one of the hypergraph edges. Figure 1(b) shows some possible relationships among instance values for the hyperedges in Figure 1(a). For example, two of the relationships are "retailer r_1 sells item i_1 for \$3" and "manufacturer m_1 has factory f_1." Figures 2(a), 2(b), and 2(c) show three possible sets of scheme trees and their associated instances taken from the relationships in Figure 1(b). In Figure 2(a), because there is only one scheme-tree instance, the data values are compactly stored. However, the instance data is redundant. Since manufacturer m_1 is necessarily stored twice, the dependent factories, which must be the same, are therefore redundantly stored more than once. In Figure 2(b), even though no data redundancy is present in any of the scheme-tree instances, there are more trees than necessary. The largest redundancy-free scheme tree for this example is the one on the left in Figure 2(c), which balances the requirements of data redundancy and compactness of data. Creating this scheme tree first followed by creating a scheme tree from the remaining hyperedge {Manufacturer, Factory} yields the fewest possible redundancy-free scheme trees. □

By way of comparison with the XML normalization work of others [2, 6, 15, 22, 25, 26, 27], 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 the complicated FDs defined in these papers, we rely on standard FD and hypergraph-generated MVD definitions, 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.

We give the details of our contribution of generating a largest possible scheme tree from a conceptual-model hypergraph in polynomial time as follows. We first lay the ground work by providing basic definitions in Section 2. Based on this foundation, we present the polynomial-time, scheme-tree generation algorithm in Section 3. Throughout Sections 2 and 3 we provide examples

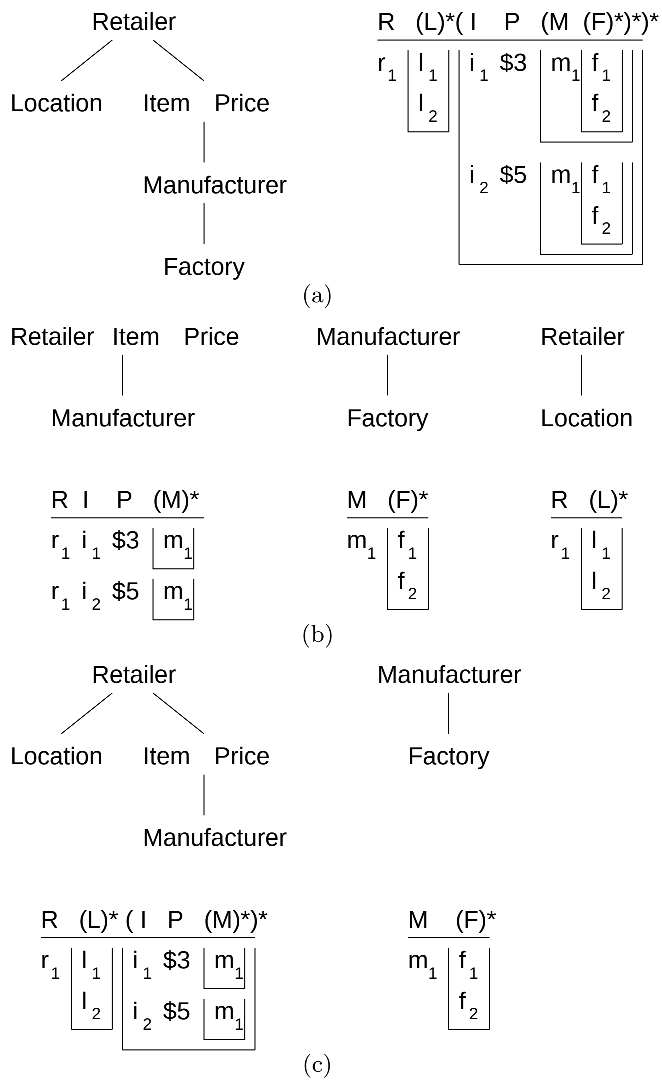


Figure 2: The Scheme Trees and Scheme-Tree Instances of Example 1.

to motivate and illustrate definitions and algorithmic procedures. We present experimental data to verify our algorithm in Section 4 and formally prove our claims in Section 5. We make concluding remarks in Section 6.

2 Basic Definitions

2.1 Acyclic Hypergraphs

To make this paper self-contained, we borrow some definitions from previous work. The first three definitions are from [3].

Definition 1 Let U be a set of attributes. A *hypergraph* $H = \{E_1, \ldots, E_n\}$ over U is a set of subsets of U where each subset E_i is called a *hyperedge* of H, or simply an *edge* of H if the context is clear. □

Definition 2 *Graham Reduction*, also known as *GYO Reduction* [10], applies two operations to a hypergraph $H = \{E_1, \ldots, E_n\}$ ($n \geq 1$) until neither can be applied. These two operations are: (Attribute Removal) If A is an attribute that appears in exactly one edge E_i, then delete A from E_i. (Edge Removal) Delete an edge E_i if there is an edge E_j such that $i \neq j$ and $E_i \subseteq E_j$. □

Definition 3 A hypergraph is *acyclic* if Graham Reduction reduces it to the empty set. □

Definition 4 A hypergraph is *reduced* if none of its hyperedges is a proper subset of another hyperedge.

By repeatedly applying the edge-removal step of Graham Reduction, it is easy to observe that a hypergraph is acyclic if and only if its reduced form is acyclic. All hypergraphs considered in this paper are assumed to be reduced.

We now introduce a procedure that makes use of Graham Reduction to create a data structure from a reduced acyclic hypergraph called a *join tree*.

Procedure `CreateJoinTree`
Input: a reduced acyclic hypergraph H.
Output: a join tree T for H, and a set of labels for H.
1. Initially, let T be a graph with no edges whose nodes are the unique hyperedges in H.
2. Apply Graham Reduction: while applying Graham Reduction, when a remaining hyperedge E_i', which is the result of applying one or more attribute removals to an original hyperedge E_i, is removed because it is a subset of an original hyperedge E_j, create an edge $\{E_i, E_j\}$ for T and label the edge E_i'. In the process, E_i' becomes a label of H. (Since E_i' may be a subset of more than one hyperedge, more than one join tree is possible for a given reduced acyclic hypergraph.)
3. When the Graham Reduction is complete, the graph T will have become a join tree; thus return T. □

Example 2 Figure 3 shows a possible join tree created by Procedure `CreateJoinTree` for the acyclic hypergraph in Figure 1(a). In another join tree for the hypergraph in Figure 1(a), instead of the *Retailer* edge between {*Retailer*, *Location*} and {*Retailer*, *Item*, *Price*}, the join tree can have a *Retailer* edge between {*Retailer*, *Location*} and {*Retailer*, *Item*, *Manufacturer*}. □

2.2 Constraints

In this paper, FDs and hypergraph-generated MVDs are the only constraints we consider. These are typically the most common constraints encountered in practice. FDs have their standard definition. The definition of hypergraph-generated MVDs is from [3] and [8].

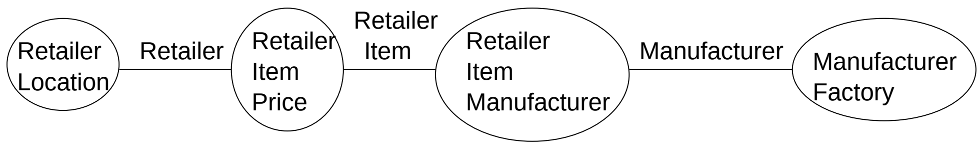


Figure 3: A Join Tree of the Acyclic Hypergraph in Figure 1(a).

Definition 5 Two hyperedges are *connected* if they have a nonempty intersection. A set S of hyperedges is *disconnected* if S can be partitioned into two nonempty subsets S_1 and S_2 such that no hyperedge in S_1 is connected to any hyperedge in S_2. A set of hyperedges is *connected* if it is not disconnected. A *connected component* is a maximal connected set of hyperedges. A hypergraph H *generates* a number of MVDs of the form $X \twoheadrightarrow Y_1|Y_2|\cdots|Y_n$ where X and $Y_1, \ldots, Y_n$ are disjoint sets of attributes and each Y_i is a maximal connected set of hyperedges constructed from the hyperedges of H after they have been reduced by the removal of the attributes in X, i.e., the maximal connected components of $\{E - X : E \text{ is a hyperedge of } H\} - \{\emptyset\}$. □

Example 3 Removing the attributes *Retailer* and *Item* from Figure 1(a) results in the hypergraph-generated MVDs *Retailer Item* $\twoheadrightarrow$ *Manufacturer Factory* | *Price* | *Location*. Removing *Manufacturer* and *Factory* results in the trivial MVD *Manufacturer Factory* $\twoheadrightarrow$ *Retailer Item Price Location*. □

2.3 Nested Normal Form (NNF)

To help achieve our goal, we make use of NNF [19] in this paper. We have proved in [19] that a scheme tree does not permit redundancy with respect to a set of MVDs and FDs if and only if it is in NNF. Thus, our goal in this paper is to extract a largest NNF scheme tree.

Definition 6 A *scheme tree* T over a set U of attributes is a rooted tree in which every node is a nonempty subset of U. Further, the intersection of every pair of nodes in T is empty. □

Definition 7 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. A *scheme-tree instance* over T is recursively defined as follows:

1. If T has only the root node $A_1 \cdots A_n$ ($n \geq 1$), a *scheme-tree instance* over T is a (possibly empty) set of functions $\{t_1, \ldots, t_m\}$ such that each t_i ($1 \leq i \leq m$) maps each A_j ($1 \leq j \leq n$) to a value in $dom(A_j)$.

2. If T has more than one node, then let $T_1, \ldots, T_k$ ($k \geq 1$) be the k subtrees of T such that the root node of each T_i is a child node of T's root node. Let $\{t_1, \ldots, t_m\}$ ($m \geq 0$) be the set of functions associated with T's root node and let $t_j \oplus s_{j_i}$ mean that the function t_j associates with the scheme-tree instance s_{j_i} over T_i for t_j. Then, $\cup_{j=1}^{m}(\cup_{i=1}^{n} t_j \oplus s_{j_i})$ is a *scheme-tree instance* over T. □

Although formally defined in Definition 7, scheme-tree instances are most easily understood when visualized and written as are the scheme-tree instances in Figure 2. In Figure 2, we nest attribute names in parentheses in a linear fashion according to their structure and place instance values in buckets (with the outermost bucket omitted).

Let T be a scheme tree. We denote the set of attributes in T by $Aset(T)$. Let N be a node in T. Notationally, $Ancestor(N)$ denotes the union of attributes in all ancestors of N, including N. Similarly, $Descendent(N)$ denotes the union of attributes in all descendants of N, including N. In a scheme tree T, each edge (V, W), where V is the parent of W, denotes an MVD $Ancestor(V) \twoheadrightarrow Descendent(W)$. 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 scheme-tree instance for T. Since FDs are also of interest, we use $FD(T)$ to denote the set of FDs that hold in T.

Example 4 Figures 2(a), 2(b), and 2(c) show three possible sets of scheme trees and their instances derived from the data in Figure 1(b). As in [19] we use a repeating-group $(\ldots)^*$ to denote a nested scheme tree and a bucket to denote a nested scheme-tree instance. Let T be the left scheme tree in Figure 2(c). Each edge in T implies an MVD. Therefore, $MVD(T)$ is equal to $\{Retailer \twoheadrightarrow Location, Retailer \twoheadrightarrow Item\ Price\ Manufacturer, Retailer\ Item\ Price \twoheadrightarrow Manufacturer\}$. In addition, $FD(T)$ is equal to $\{Retailer\ Item \rightarrow Price\}$ as declared in Figure 1(a). $\square$

Definition 8 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 $Aset(T) \subseteq U$. T is in NNF with respect to $M \cup F$ if the following conditions are satisfied.

1. Let D be the set of MVDs and FDs that hold for T with respect to $M \cup F$. The set D is equivalent to $MVD(T) \cup FD(T)$ on $Aset(T)$.

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

Example 5 All scheme trees in Figures 2(b) and 2(c) are in NNF. The scheme tree in Figure 2(a), however, is not in NNF. To see this, let T be the scheme tree in Figure 2(a). Then, $MVD(T) = \{Retailer \twoheadrightarrow Location, Retailer \twoheadrightarrow Item\ Price\ Manufacturer\ Factory, Retailer\ Item\ Price \twoheadrightarrow Manufacturer\ Factory, Retailer\ Item\ Price\ Manufacturer \twoheadrightarrow Factory\}$, and $FD(T) = \{Retailer\ Item \rightarrow Price\}$. Now, observe that $Manufacturer \twoheadrightarrow Factory$ is a hypergraph-generated MVD that holds in T (obtained by removing *Manufacturer* from the hypergraph in Figure 1(a)). Using the chase [16], it is easy to show that $MVD(T) \cup FD(T)$ does not imply $Manufacturer \twoheadrightarrow Factory$ and therefore that T violates NNF's Condition 1. $\square$

2.4 Syntactic Covers

Syntactic covers guarantee that every value and every relationship in an associated instance of a hypergraph can appear in a scheme-tree instance (e.g., that the values and relationships in the instance in Figure 1(b) can appear in the scheme tree instances in Figure 2.) Since we are generating storage structures, syntactic coverage is a necessary condition for any set of scheme trees generated for a hypergraph.

In the following, for any subset S of a hypergraph H, we use the notation $\overline{S}$ to denote the set $\cup_{E_i \in S} E_i$. $\overline{S}$ is simply the set of attributes in some set of hypergraph edges.

Definition 9 A *path* of a scheme tree T is a sequence of nodes from the root node of T to a leaf node of T. Let H be a hypergraph. An attribute $A \in \overline{H}$ *appears* in a scheme tree T if A is in a node of T. A hyperedge $E \in H$ *appears* in a scheme tree T if there is a path in T whose nodes collectively contain all of E's attributes.[2] $\Box$

Definition 10 A scheme tree T *syntactically covers* a set S of hyperedges if (1) $Aset(T) = \overline{S}$, and (2) every hyperedge in S appears in a path of T. A scheme-tree forest F *syntactically covers* a hypergraph H if there are subsets $S_1, \ldots, S_n$ of hyperedges in H such that $\overline{S_1} \cup \cdots \cup \overline{S_n} = \overline{H}$ and there are scheme trees $T_1, \ldots, T_n$ in F such that T_i syntactically covers S_i $(1 \leq i \leq n)$. $\Box$

Example 6 All three sets of scheme trees in Figures 2(a), 2(b) and 2(c) syntactically cover the hypergraph in Figure 1(a). As an example of failure to syntactically cover, consider the first scheme tree in the scheme-tree forest in Figure 2(c). If we remove *Price*, there is no place for *Price* values. Clearly, every attribute must appear in the scheme-tree forest. If we remove *Manufacturer*, although there is still a place for *Manufacturer* values in the second scheme tree in Figure 2(c), there is no place for the triples that belong to the edge $\{Retailer,\ Item,\ Price\}$. Clearly, every edge must appear in a path of some scheme tree. $\Box$

3 Extracting a Largest NNF Scheme Tree

The main algorithm of this paper extracts a largest NNF scheme tree from a reduced acyclic hypergraph and a set F of embedded FDs such that each hyperedge is in BCNF. The algorithm calls several procedures, which are explained in detail in the following sections. As a summary, Step 1 reduces the number of input hyperedges. Step 2 creates a join tree and a set of labels for the acyclic hypergraph. Step 3 constructs a Hasse diagram of a partial order defined on the acyclic hypergraph's labels. Step 4 refines the join tree created in Step 2. Step 5 extracts a largest NNF skeleton from the Hasse diagram. Finally, Step 6 attaches the NNF skeleton's hyperedges to the skeleton to make it a largest NNF scheme tree.

[2] Note that the definition of syntactic coverage for this paper differs from the definition in [18]. In [18] the definition requires a hyperedge to appear in contiguous nodes in a path of a scheme tree while the definition here does not. Since we make the universal relation assumption in this paper and we did not for [18], we can relax the condition of syntactic coverage in [18]. For example, consider a reduced, acyclic hypergraph $H = \{AV_1,\ ABV_2,\ ABCV_3,\ ACV_4\}$ and an embedded FD $AC \to B$. A NNF scheme tree T for H has A as the root node, A's child nodes are B and V_1, B's child nodes are C and V_2, and C's child nodes are V_3 and V_4. The hyperedge ACV_4 does not appear in contiguous nodes in any path in T. Nevertheless, T is in NNF and T syntactically covers the entire hypergraph under the definition of syntactic coverage of this paper.

3.1 The Main Algorithm

Procedure `Main`

Input: a reduced acyclic hypergraph H and a set F of embedded FDs such that each hyperedge in H is in BCNF.

Output: a largest NNF scheme tree.

1. Call Procedure `MergeHyperedges`.
2. Call Procedure `CreateJoinTree`.
3. Call Procedure `ConstructHasseDiagramOf`$\succeq$.
4. Call Procedure `MoveLabelsToCenterNodes`.
5. Call Procedure `ExtractLargestNNFSkeleton`.
6. Call Procedure `AttachHyperedges`. $\Box$

3.2 Procedure `MergeHyperedges`

Two distinct hyperedges E_i and E_j are *functionally equivalent* if $E_i \to E_j$ and $E_j \to E_i$. Theorem 1 of Section 5.1 states that there is no loss of generality to assume that no two distinct functionally equivalent hyperedges exist. Hence, Procedure `MergeHyperedges` merges functionally equivalent hyperedges together to reduce the number of input hyperedges. From now on, we can safely assume that no two distinct functionally equivalent hyperedges exist.

Procedure `MergeHyperedges`

Input: a reduced acyclic hypergraph H and a set F of embedded FDs such that each hyperedge in H is in BCNF.

Output: a reduced acyclic hypergraph H with no distinct functionally equivalent hyperedges and the same set F of embedded FDs.

1. Call Algorithm 4.4 on page 66 in [16] to compute E^+ for each hyperedge $E \in H$.
2. Put hyperedges E_i and E_j in the same set if $E_i^+ = E_j^+$.
3. For each set S with two or more hyperedges, do:
   Merge all hyperedges in S together to form a new hyperedge and add it to H.
   Remove each hyperedge in S from H. $\Box$

Example 7 Consider the FDs and hyperedges in Figure 4(a).[3] Every FD is embedded in some hyperedge, every hyperedge is in BCNF, and these hyperedges together constitute an acyclic hypergraph. The hyperedges AV_2 and AV_3 in Figure 4(a) are functionally equivalent because $AV_2^+ = AV_3^+ = AV_2V_3$. Thus, Procedure `MergeHyperedges` merges AV_2 and AV_3 together to form a new hyperedge AV_2V_3 and removes AV_2 and AV_3 from H. A join tree created by Procedure `CreateJoinTree` in Section 2.1 for the resulting acyclic hypergraph is shown in Figure 4(b). $\Box$

[3] $V_1, \ldots, V_{16}$ are attributes that appear in exactly one hyperedge. Attributes that appear in exactly one hyperedge are not essential for our algorithm.

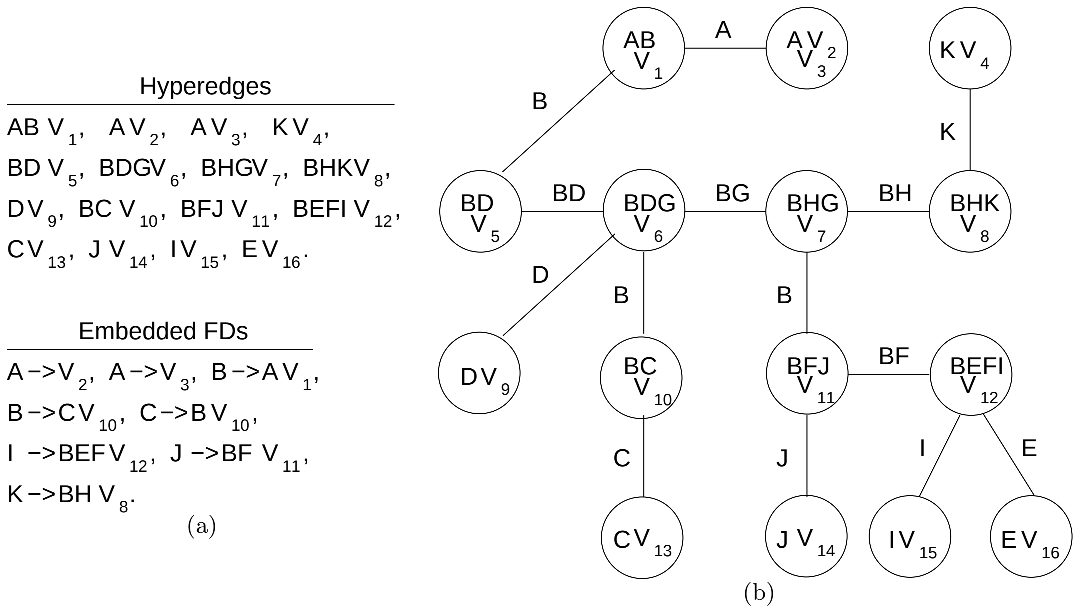


Figure 4: Merging Functionally Equivalent Hyperedges and Creating a Join Tree.

3.3 Procedure `ConstructHasseDiagramOf`$\succeq$

We now define a partial order on the labels of the input reduced acyclic hypergraph. Later we derive a largest NNF scheme tree from the Hasse diagram of this partial order.

Definition 11 Let H be a reduced acyclic hypergraph and F be a set of embedded FDs. Two distinct labels L_i and L_j of H are *functionally equivalent* if $L_i \rightarrow L_j$ and $L_j \rightarrow L_i$. Let $C_1, \ldots, C_n$ be the equivalence classes[4] of labels of H such that all the labels in each equivalence class C_i are pairwise functionally equivalent. We define $\succeq$ to be a partial order on $C_1, \ldots, C_n$ in which $C_i \succeq C_j$ if $L_i \rightarrow L_j$ where $L_i \in C_i$ and $L_j \in C_j$. $\Box$

Lemma 6 of Section 5.3 states that the multiset of labels in any join tree for an acyclic hypergraph is the same. Therefore, the partial order $\succeq$ and its derived Hasse diagram are unique for the input reduced acyclic hypergraph and the embedded FDs.

Procedure `ConstructHasseDiagramOf`$\succeq$
Input: a join tree J and a set F of embedded FDs such that each node in J is in BCNF.
Output: the Hasse diagram of $\succeq$.
1. Call Algorithm 4.4 on page 66 in [16] to compute L^+ for each label L of J.
2. Put labels L_i and L_j in the same equivalence class if $L_i^+ = L_j^+$.
3. For two equivalence classes C_i and C_j, $C_i \succeq C_j$ if $L_i^+ \supseteq L_j^+$ where $L_i \in C_i$ and

[4]An equivalence class is a set, not a multiset.

$L_j \in C_j$.

4. Generate the Hasse diagram of $\succeq$. $\Box$

Example 8 The labels B and C in Figure 4(b) are in the same equivalence class because they are functionally equivalent. On the other hand, each of the other labels in Figure 4(b) is in a different equivalence class. The Hasse diagram of $\succeq$ is shown in Figure 5(a), in which $\{BD\} \succeq \{B, C\}$, $\{BD\} \succeq \{D\}$, and $\{B, C\} \succeq \{A\}$, and so on. $\Box$

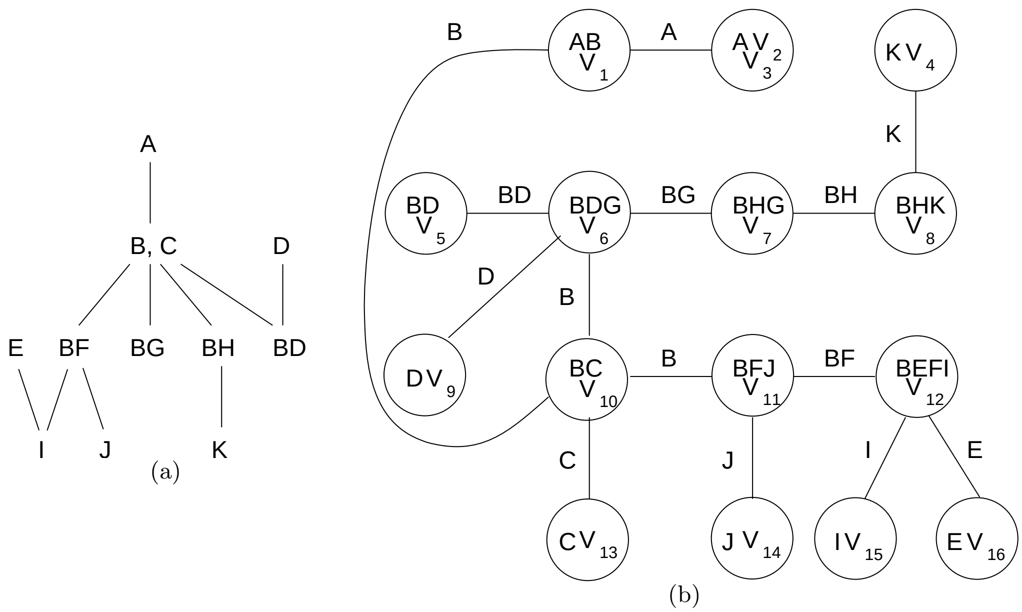


Figure 5: Constructing the Hasse diagram of $\succeq$ and Moving Labels to Center Nodes.

3.4 Procedure `MoveLabelsToCenterNodes`

Lemmas 7 and 8 of Section 5.3 together state that all distinct labels of any equivalence class of labels are incident with a unique common node in a join tree. We call such a node the *center node* of the equivalence class. Procedure `MoveLabelsToCenterNodes` makes all labels in a join tree that appear in an equivalence class incident with the equivalence class's center node.

Procedure `MoveLabelsToCenterNodes`

Input: a join tree J and a set of equivalence classes of labels in J.

Output: a modified join tree J with all labels in J that appear in an equivalence class of labels incident with the equivalence class's center node.

1. For each equivalence class C with two or more distinct labels, do:

    Locate the center node E of C.

    For each edge $\{E_i, E_j\}$ in J such that $E_i \cap E_j$ is a label in C, do:

        Remove $\{E_i, E_j\}$ from J.

If E_i becomes disconnected from E, then

establish an edge $\{E_i,\, E\}$ with the label $E_i \cap E_j$.

Else

establish an edge $\{E_j,\, E\}$ with the label $E_i \cap E_j$.

2. For each equivalence class C with exactly one label, do:

   Arbitrarily choose one node of an edge in J with that label.

   Designate that node as the center node for C.

   Repeat the inner for-loop in Step 1. □

Example 9 Since $B \rightarrow C$ and $C \rightarrow B$, we have the equivalence class of labels $\{B,\, C\}$. The center node for $\{B, C\}$ is BCV_{10} in Figure 4(b). The result of applying Procedure `MoveLabelsToCenterNodes` on the join tree in Figure 4(b) is shown in Figure 5(b). □

3.5 Procedure `ExtractLargestNNFSkeleton`

Theorem 2 of Section 5.2 states that if an NNF scheme tree syntactically covers some hyperedges, the hyperedges must be the nodes in a connected subtree of a join tree. Additionally, Theorem 3 of Section 5.3 states that to satisfy NNF, this connected subtree cannot have any critical node. Based on these two theorems, creating a largest NNF scheme tree that contains the greatest number of hyperedges is the same as creating a largest NNF scheme tree that syntactically covers the nodes in a connected subtree of a join tree where (1) the number of nodes in the connected subtree is the greatest and (2) the connected subtree has no critical nodes. To accomplish this goal, we first find a largest NNF skeleton in the Hasse diagram of $\succeq$ that contains the greatest number of labels. Then, we attach the hyperedges with which these labels are incident to this skeleton to make it a largest NNF scheme tree. The definitions of these concepts now follow.

Definition 12 A *connected subtree* of a join tree T is inductively defined as follows: (1) A single node in T is a *connected subtree* of T. (2) If N' is a node in a connected subtree T' of T and N is a node in T such that $\{N,\, N'\}$ is an edge in T, then T' augmented with the node N and the edge $\{N,\, N'\}$ is a *connected subtree* of T. Let T' be a connected subtree of a join tree. The notation $\overline{T'}$ denotes the union of all the hyperedges that are nodes in T'. □

Definition 13 Let H be an acyclic hypergraph and F be a set of embedded FDs in H. Let J be a join tree for H and S be a connected subtree of J, which is not necessarily a proper subset. A label L of H *belongs* to S if there is an edge E in S such that E's label is L. A node N in S is *critical* with respect to S if there are two labels L_i and L_j belonging to S such that $L_i \not\rightarrow L_j$, $L_j \not\rightarrow L_i$, and $(L_i \cup L_j) \subseteq N$. If S is actually J, then we may simply call a node of J critical without having to make any reference to S. □

Definition 14 Given an equivalence class C in the Hasse diagram of $\succeq$, any tree rooted at C extracted from the Hasse diagram of $\succeq$ is called a *skeleton*. Let K be a skeleton and J be a join tree. K's *induced set of edges* is the set $\{E$ is an edge in J : E's label appears in an equivalence class in $K\}$. A *NNF skeleton* is a skeleton whose induced set of edges constitutes a connected subtree of J and the connected subtree has no critical nodes. □

Definition 15 Let C_i and C_k be two equivalence classes of labels such that C_i is a parent node of C_k in the Hasse diagram of $\succeq$. C_k is a *nontrivial child* of C_i, or $C_k \succeq C_i$ *nontrivially*, if for each $L_i \in C_i$ and for each $L_k \in C_k$, $L_i \not\subseteq L_k$. On the other hand, if there are labels $L_i \in C_i$ and $L_k \in C_k$ such that $L_i \subset L_k$, then C_k is a *trivial child* of C_i, or $C_k \succeq C_i$ *trivially*. □

Theorem 4 of Section 5.4 proves that Procedure `ExtractLargestNNFSkeleton` indeed outputs NNF skeletons.

Procedure `ExtractLargestNNFSkeleton`
Input: the Hasse diagram of $\succeq$ and the modified join tree J.
Output: a largest NNF skeleton.
1. For each equivalence class C of labels in the Hasse diagram of $\succeq$, do:
    Associate with C an integer variable *labelCnt* and set $C.labelCnt = 0$.
    Associate with C a set of edges in J called *myEdges* where $C.myEdges =$
    $\{E$ is an edge in J : E's label appears in $C\}$.
2. For each root node R in the Hasse diagram of $\succeq$, do:
    Call Procedure `CalculateLabelCnt`(R).
3. Select a root node R in the Hasse diagram of $\succeq$ with the greatest *labelCnt*.
4. Return the NNF skeleton rooted at R. □

Procedure `CalculateLabelCnt`(C: an equivalence class in the Hasse diagram of $\succeq$)
1. If $C.labelCnt = 0$, then
    For each nontrivial child D of C, do:
        Call Procedure `CalculateLabelCnt`(D).
        $C.labelCnt = C.labelCnt + D.labelCnt$.
    For each trivial child D of C, do:
        Call Procedure `CalculateLabelCnt`(D).
    While there is an unmarked trivial child of C, do:
        Set *maxD* to an unmarked trivial child of C with the greatest *labelCnt*.
        Mark *maxD*.
        For each other unmarked trivial child D of C, do:
            If the path between D's center node and *maxD*'s center node
            in J does not contain any edge in $C.myEdges$, then
                Remove D as a trivial child of C.
    For each marked trivial child D of C, do:
        $C.labelCnt = C.labelCnt + D.labelCnt$.
    $C.labelCnt = C.labelCnt +$ the size of $C.myEdges$. □

Example 10 Steps 1, 3, and 4 of Procedure `ExtractLargestNNFSkeleton` are straightforward. Let us focus on Step 2. With respect to the Hasse diagram in Figure 5(a), there are three root nodes, namely $\{E\}$, $\{A\}$, and $\{D\}$. Suppose the root node R in Step 2 is $\{A\}$. Initially, $\{A\}.labelCnt = 0$. Thus, Procedure `CalculateLabelCnt` enters the if-statement. Then, Procedure `CalculateLabelCnt` recursively calls itself until it reaches the leaf nodes of the Hasse diagram. The five leaf nodes in the Hasse diagram are $\{I\}$, $\{J\}$, $\{BG\}$, $\{K\}$, and $\{BD\}$. Since none of these equivalence classes of labels has a child and since there is only one label in Figure 5(b) that appears in each of these equivalence classes, $\{I\}.labelCnt = 1$, $\{J\}.labelCnt = 1$, $\{BG\}.labelCnt = 1$, $\{K\}.labelCnt = 1$, and $\{BD\}.labelCnt = 1$. As Procedure `CalculateLabelCnt` unwinds from recursion, since $\{I\}$ and $\{J\}$ are nontrivial children of $\{BF\}$, $\{I\}.labelCnt$ and $\{J\}.labelCnt$ are added to $\{BF\}.labelCnt$. And because there is only one label in Figure 5(b) that appears in $\{BF\}$, $\{BF\}.labelCnt = 3$, as Figure 6(a) shows. The same reasoning applies to $\{BH\}.labelCnt$. Now, Procedure `CalculateLabelCnt` unwinds to the equivalence class $\{B, C\}$ in Figure 5(a). $\{BF\}$ is selected first because $\{BF\}.labelCnt$ is the greatest among all $\{B, C\}$'s trivial children. However, there is a label B in between of $\{BF\}$'s center node and the center node of the other trivial child of $\{B, C\}$ in Figure 5(b). Thus, the inner for-loop of the while-loop in Procedure `CalculateLabelCnt` has no effect. The next trivial child to be selected is $\{BH\}$. However, with respect to Figure 5(b), neither the path between $\{BH\}$'s center node and $\{BG\}$'s center node nor the path between $\{BH\}$'s center node and $\{BD\}$'s center node contains any label B or label C. Thus, Procedure `CalculateLabelCnt` removes $\{BG\}$ and $\{BD\}$ as trivial children of $\{B, C\}$. There are four labels in Figure 5(b) that appear in $\{B, C\}$. Thus, $\{B, C\}.labelCnt = \{BF\}.labelCnt + \{BH\}.labelCnt + 4 = 3 + 2 + 4 = 9$. Finally, Procedure `CalculateLabelCnt` unwinds back to $\{A\}$ and there is only one label in Figure 5(b) that appears in $\{A\}$. Thus, $\{A\}.labelCnt = \{B, C\}.labelCnt + 1 = 9 + 1 = 10$, as Figure 6(b) shows. $\{E\}.labelCnt$ and $\{D\}.labelCnt$ are calculated similarly. $\square$

3.6 Procedure `AttachHyperedges`

The last step of our algorithm attaches hyperedges to a largest NNF skeleton.

Procedure `AttachHyperedges`
Input: a largest NNF skeleton T and the modified join tree J.
Output: a largest NNF scheme tree.
1. Let S be T's induced set of edges in J.
2. For each node E in S, do:
    Find the lowest node N in T such that N contains a label L where $L \subseteq E$.
    Let $N_E = \{A \in E : A$ does not appear in any label in any equivalence class of $T\}$.
    If $N_E \neq \emptyset$, then
        Add N_E as a child node to N in T.
3. For each node N in T, do:

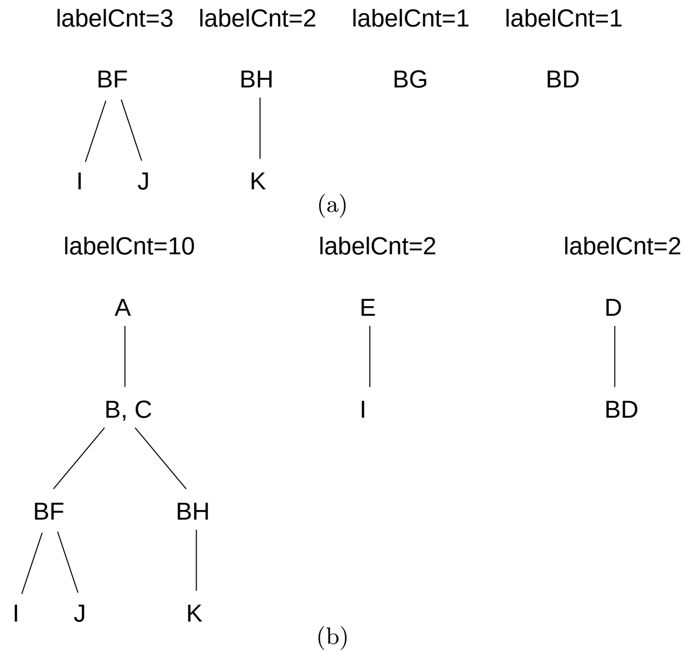


Figure 6: Calculating *labelCnt* for each Equivalence Class in the Hasse diagram of $\succeq$.

Merge all labels in N together.

4. For each child node N in T, do:

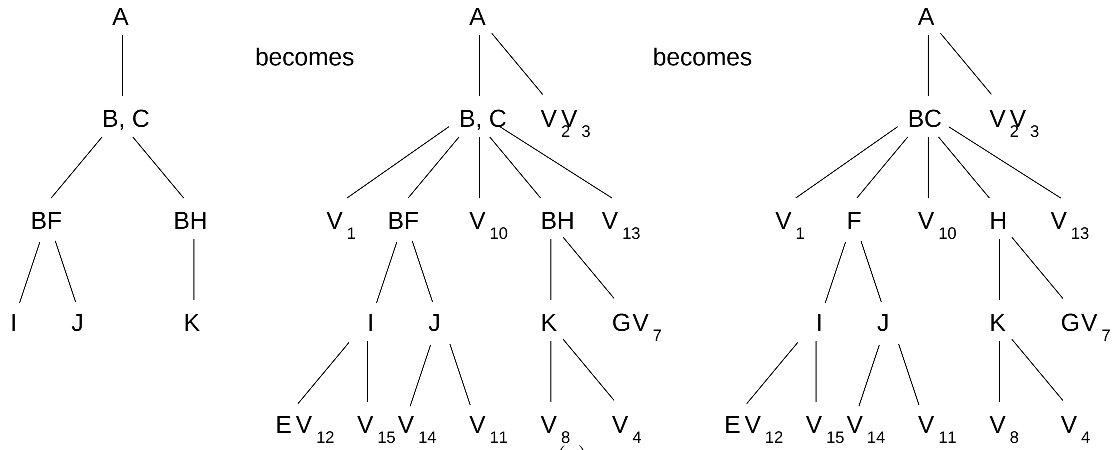
    If an attribute $A \in N$ is also in N's parent node, then

        Remove A from N. $\Box$

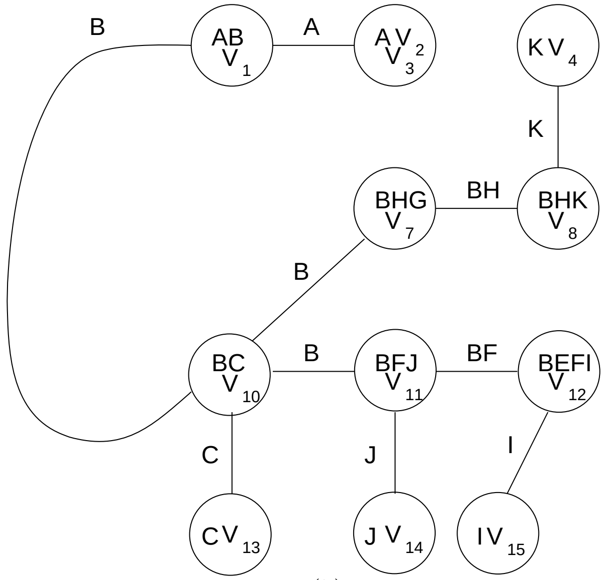
Example 11 Figure 7(a) shows how Procedure `AttachHyperedges` turns an NNF skeleton into an NNF scheme tree. As an example, the lowest node N is $\{B, C\}$ for the hyperedge ABV_1. Thus, V_1 is added as a child node to $\{B, C\}$. Then, Procedure `AttachHyperedges` merges all labels together in every node and removes every redundant attribute. Figure 7(b) shows the connected subtree defined by the set S from which the NNF scheme tree in Figure 7(a) is constructed. $\Box$

4 Experimental Evaluation

As Theorem 4 of Section 5.4 asserts, the algorithms that underlie Procedure `MergeHyperedges` and Procedure `CreateJoinTree` have been well-studied in the literature and have been proved to run in time polynomial in the size of the input. In addition, Theorem 4 also shows that Procedure `AttachHyperedges` runs in time linear with respect to the size of the input. Thus, in our experiments, we focus on Procedure `ConstructHasseDiagramOf`$\succeq$, Procedure `MoveLabelsToCenterNodes`, and Procedure `ExtractLargestNNFSkeleton`. We have implemented these procedures in a Visual Basic 2008 program, which first randomly generates join trees and equivalence classes of labels and then extracts largest NNF skeletons from them. The computer used in our experiments is a Dell



(a)



(b)

Figure 7: Turning an NNF Skeleton to an NNF Scheme Tree.

desktop PC with an E6300 Intel Core 2 CPU running at 1.86 and 1.87 GHz with 2045 MBs of memory. The operating system is Windows Vista Business Edition.

Figure 8 shows the time taken by the simulation program, measured in milliseconds (ms), when there are 2500, 5000, 7500, and 10000 hyperedges. Although the definition of a join tree does not require a root node, having a root node in a join tree makes our implementation much easier. As a result, the terms “parent nodes” and “child nodes” are applicable to our join trees. In our experiments, each internal node of a join tree randomly has 1 or any number up to $maxFanout$ child nodes, where $maxFanout$ is a variable that is set to 1, 3, 5, 10, 15, 20, 25, 50, 100 or 200. By increasing the value of $maxFanout$, more labels are clustered into an equivalence class of labels. This in turn reduces their number. Fewer equivalence classes of labels results in fewer comparisons needed to construct the partial order $\succeq$. Thus, the program takes less time to complete.

However, Figure 8 also shows that the time needed to complete the program levels off as the value of $maxFanout$ increases. One of the reasons for this phenomenon is that overhead operations take more time as the value of $maxFanout$ increases, which cancels out the advantage of increasing $maxFanout$.

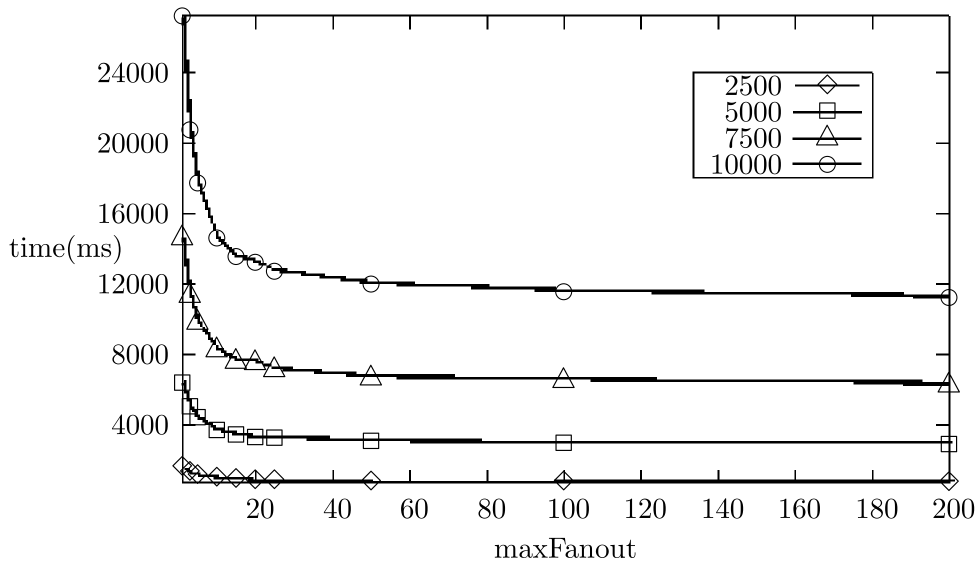


Figure 8: Plotting time against $maxFanout$.

Another observation about Figure 8 is that for each of the 10 values of $maxFanout$, doubling the number of hyperedges quadruples the time needed to find a largest NNF skeleton. This gives a hint that these three procedures as a whole run in time polynomial in the size of the input.

Figure 9 provides further evidence for this claim. For each of the 10 values of $maxFanout$, we plot n^2/time against n, where n is the number of hyperedges. In all four values of n, the ratio between n^2 and time (i.e., n^2/time) is relatively stable for a fixed value of $maxFanout$. Since our join trees and equivalence classes of labels are randomly generated, the results in Figure 9 suggest that given that all other conditions remain the same, on average Procedure `ConstructHasseDiagramOf`$\succeq$, Procedure `MoveLabelsToCenterNodes`, and Procedure `ExtractLargestNNFSkeleton` considered as a whole run in quadratic time in the size of the input.

5 Proofs for Claims

5.1 Acyclic Hypergraphs and Functionally Equivalent Hyperedges

Theorem 1, the main result of this section, states that there is no loss of generality to assume that no two distinct functionally equivalent hyperedges exist. However, before we can prove Theorem 1, we need to prove several lemmas.

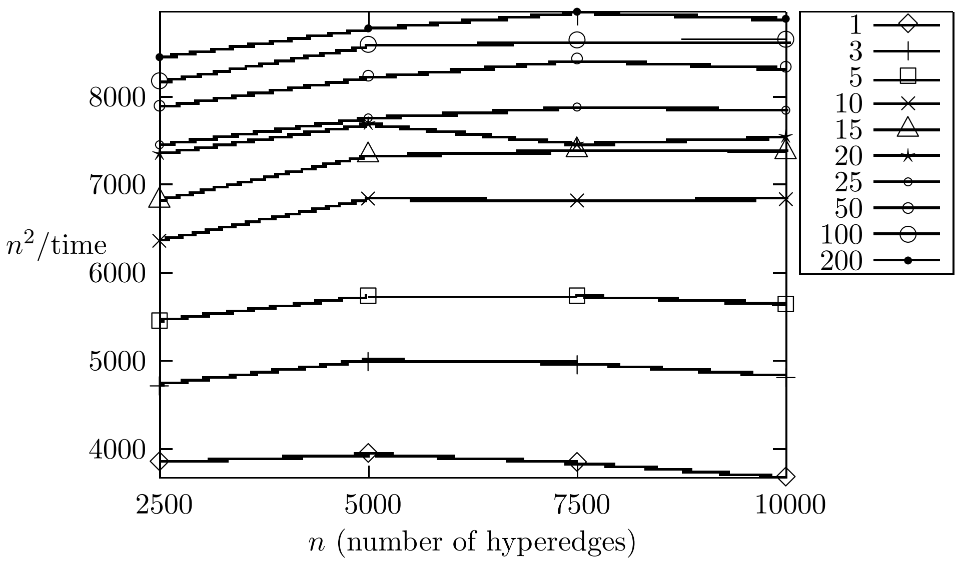


Figure 9: Plotting n^2/time against n (number of hyperedges).

Lemma 1 In a join tree T for a reduced, acyclic hypergraph, for any two distinct hyperedges E_i and E_j and for every attribute A in $E_i \cap E_j$, the label of each edge along the unique path between E_i and E_j in T contains A.
Proof. See [3]. $\square$

Let J be a join tree for an acyclic hypergraph H and $\{E_i,\, E_j\}$ be an edge in J. We use J_i to denote the connected subtree of J that contains the node E_i if the edge $\{E_i,\, E_j\}$ were removed from J. Likewise, J_j denotes the connected subtree of J that contains the node E_j if the edge $\{E_i,\, E_j\}$ were removed from J. To demonstrate how to obtain J_i and J_j from J, we may imagine cutting along the curved dashed lines in Figure 10. Let F be a set of embedded FDs in H. An FD $X \to Y \in F$ is *inside* of J_i if $XY \subseteq \overline{J}_i$;[5] otherwise, $X \to Y$ is *outside* of J_i. Note that it is possible for an FD to be inside of both J_i and J_j because $\overline{J}_i$ and $\overline{J}_j$ are not disjoint. Let W^+ be the closure of a set W of attributes. In the following, we say an FD $X \to Y \in F$ *is used in the derivation of* W^+ if $X \to Y$ is used in the second step of this process: (1) $W^+ := W$ initially; (2) $W^+ := W^+ \cup Y$ if $X \subseteq W^+$ and $Y - W^+ \neq \emptyset$.

Lemma 2 Let J be a join tree for a reduced, acyclic hypergraph H and $\{E_i,\, E_j\}$ be an edge in J. Let F be a set of embedded FDs in H. For any set W of attributes such that $W \subseteq \overline{J}_i$, if $X \to Y \in F$ is an FD that is outside of J_i and is used in the derivation of W^+, then there is a subset E_i' of E_i such that $E_i' \to Y$.
Proof. Let $X_1 \to Y_1 \in F$ be the first FD that is outside of J_i and is used in the derivation of W^+. Since the FDs used before $X_1 \to Y_1$ for generating W^+ are all inside of J_i, $X_1 \subseteq \overline{J}_i$. Since $X_1 \to Y_1$

[5]Recall that the notation $\overline{J}_i$ denotes the union of all the hyperedges that are nodes in J_i.

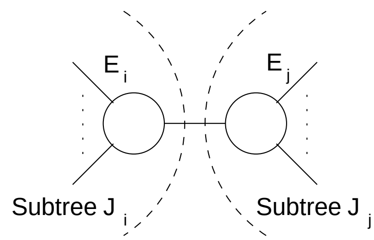


Figure 10: The Connected Subtrees J_i and J_j of Lemma 2.

is outside of J_i and $X_1 \subseteq \overline{J}_i$, by Lemma 1, it must be that $X_1 \subseteq E_i$. Thus, the basis is established. Assume the lemma is true for k ($k \geq 1$) or less FDs in F that are outside of J_i and are used in the derivation of W^+. Now, consider another FD $X_{k+1} \rightarrow Y_{k+1} \in F$ that is outside of J_i and is used in the derivation of W^+. We first partition X_{k+1} into two sets: $X_{k+1} \cap \overline{J}_i$ and $X_{k+1} - \overline{J}_i$. Since $X_{k+1} \rightarrow Y_{k+1}$ is outside of J_i, by Lemma 1, $X_{k+1} \cap \overline{J}_i$ is a subset of E_i. Now, consider an attribute A in $X_{k+1} - \overline{J}_i$. Since $A \in X_{k+1}$ and $X_{k+1} \rightarrow Y_{k+1}$ is used in generating W^+, $A \in W^+$. Then, before applying $X_{k+1} \rightarrow Y_{k+1}$ it must be that A has been added to W^+ by an FD in F that is outside of J_i. By the induction hypothesis, there is a subset E'_A of E_i such that $E'_A \rightarrow A$. Hence, by forming the union of every E'_A for each A in $X_{k+1} - \overline{J}_i$ and $X_{k+1} \cap \overline{J}_i$, there is a subset E'_i of E_i such that $E'_i \rightarrow Y_{k+1}$. $\Box$

Example 12 Consider the edge with the label BG in Figure 4(b). On the different sides of this edge are the attribute J and the FD $B \rightarrow AV_1$. The attribute A is added to J^+ by $B \rightarrow AV_1$; and thus by Lemma 2, the node $BHGV_7$ has a subset, namely B, that functionally determines A (i.e., $B \rightarrow A$). $\Box$

Similar to what we have done for Lemma 2, we define some terms for Lemma 3. Let J be a join tree for an acyclic hypergraph H and J' be a connected subtree of J. Let F be a set of embedded FDs in H. An FD $X \rightarrow Y \in F$ is *inside* of J' if $XY \subseteq \overline{J'}$; otherwise, $X \rightarrow Y$ is *outside* of J'. Notationally, we let F^+ be the closure of F, $F^+[E]$ be the set $\{X \rightarrow Y \in F^+ : E \in H$ and $XY \subseteq E\}$ and $F^+[J']$ be the set $\cup_{E \in S} F^+[E]$ where S is the set $\{E \in H : E$ is a node in $J'\}$.

Lemma 3 Let J be a join tree for a reduced, acyclic hypergraph H and J' be a connected subtree of J. Let F be a set of embedded FDs in H. For any set W of attributes such that $W \subseteq \overline{J'}$, $F^+[J']$ is sufficient to derive $W^+ \cap \overline{J'}$.

Proof. Since J' is a connected subtree of J, removing the nodes (hyperedges) in J' from J will partition the remaining nodes in J into one or more connected subtrees J_1, J_2, $\ldots$, J_p ($p \geq 1$). Two of them are shown in Figure 11, in which the dashed lines outline the boundaries of J_i, J_j and J'. In addition, for each J_i ($1 \leq i \leq p$), there is a node called E_i in J' that connects directly to a node in J_i.

If we only need the FDs in F that are inside of J' to derive $W^+ \cap \overline{J'}$, this lemma is vacuously true. Hence, suppose we need some FDs in F that are outside of J' to generate $W^+ \cap \overline{J'}$. We

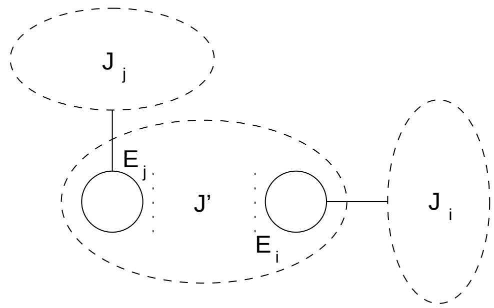


Figure 11: The Connected Subtrees J', J_i and J_j of Lemma 3.

now describe a procedure that derives $W^+ \cap \overline{J'}$ by using the FDs in $F^+[J']$. Thus, we demonstrate that $F^+[J']$ is sufficient to derive $W^+ \cap \overline{J'}$. Without loss of generality, we assume the right-hand side of each FD in F is a single attribute. For this procedure, we declare F', a set of FDs, that is continually a subset of $F^+[J']$. F' is initially set to $\{X \to A \in F : X \to A$ is inside of $J'\}$. We then apply the FDs in F' to generate W^+ until no more attribute can be added. Assume n ($n \geq 0$) such FDs are applied in this order:

$$\begin{gathered} X_1 \to A_1 \in F', \\ X_2 \to A_2 \in F', \\ \vdots \\ X_n \to A_n \in F'. \end{gathered}$$

Because these n FDs are all in F', $A_1 \in \overline{J'}$, $A_2 \in \overline{J'}$, $\ldots$, and $A_n \in \overline{J'}$. At this point, we have to apply some FDs in $F - F'$ in order to continue to add attributes. Assume m ($m \geq 2$) such FDs are applied in this order:

$$\begin{gathered} X_{n+1} \to A_{n+1} \in F - F', \\ \vdots \\ X_{n+m-1} \to A_{n+m-1} \in F - F', \\ X_{n+m} \to A_{n+m} \in F - F'. \end{gathered}$$

Without loss of generality, we assume that these m FDs are selected in such a way that $A_{n+1}, \ldots, A_{n+m-1}$ must all be added to W^+ before A_{n+m} can be added to W^+, and also $A_{n+1} \notin \overline{J'}, \ldots, A_{n+m-1} \notin \overline{J'}$, and $A_{n+m} \in \overline{J'}$. Since $(\overline{J_i} - \overline{J'}) \cap (\overline{J_j} - \overline{J'}) = \emptyset$ when $i \neq j$ ($1 \leq i, j \leq p$), our assumption implies that the FDs $X_{n+1} \to A_{n+1}, \ldots, X_{n+m-1} \to A_{n+m-1}, X_{n+m} \to A_{n+m}$ are all inside of the same connected subtree J_i. Note that since $A_{n+m} \in \overline{J'}$ and $X_{n+m} \to A_{n+m}$ is outside of J', by Lemma 1, $A_{n+m} \in E_i$.

By Lemma 2, for each $X_j \to A_j$ ($n+1 \leq j \leq n+m$), there is a subset E_{i_j} of E_i such that $E_{i_j} \to A_j$. We now show by induction that F' implies $W \to E_{i_j}$ ($n+1 \leq j \leq n+m$). For the FD $X_{n+1} \to A_{n+1}$, since $X_{n+1} \to A_{n+1}$ is outside of J' and $X_{n+1} \subseteq WA_1 \cdots A_n \subseteq \overline{J'}$, by Lemma 1, $X_{n+1} \subseteq E_i$. Thus, X_{n+1} is the subset of E_i that we want; and obviously F' implies $W \to X_{n+1}$.

Hence, the basis is established. Now, consider an FD $X_k \rightarrow A_k$ for some k where $n+1 < k \leq n+m$. With respect to the order of applying the FDs, $X_k \subseteq WA_1 \cdots A_n A_{n+1} \cdots A_{k-1}$. We now partition X_k into two sets: $X_k \cap \overline{J'}$ and $X_k - \overline{J'}$. Since $A_{n+1}, A_{n+2}, \ldots, A_{k-1}$ are not in $\overline{J'}$, therefore the argument for $X_k \cap \overline{J'}$ is the same as that for X_{n+1}. That is, $X_k \cap \overline{J'} \subseteq WA_1 \cdots A_n \subseteq \overline{J'}$, $X_k \cap \overline{J'} \subseteq E_i$ and F' implies $W \rightarrow X_k \cap \overline{J'}$. Now, consider an attribute $A \in X_k - \overline{J'}$. Since $A \in X_k$ and $A \notin \overline{J'}$, A must be added to W^+ by an FD before $X_k \rightarrow A_k$ in the above order. By Lemma 2, there is a subset E_A of E_i such that $E_A \rightarrow A$; and by the induction hypothesis, F' implies $W \rightarrow E_A$. Thus, if we let S be the union of every E_A for each attribute $A \in X_k - \overline{J'}$, then $S \cup (X_k \cap \overline{J'})$ is a subset of E_i, F' implies $W \rightarrow S \cup (X_k \cap \overline{J'})$, $S \cup (X_k \cap \overline{J'}) \rightarrow X_k$ and $S \cup (X_k \cap \overline{J'}) \rightarrow Y_k$. This means $S \cup (X_k \cap \overline{J'})$ is the subset of E_i that we want[6] and our induction step is complete. Now, by setting $k = n+m$, we have F' implies $W \rightarrow E_{i_{n+m}}$ where $E_{i_{n+m}} \subseteq E_i$ and $E_{i_{n+m}} \rightarrow A_{n+m}$. Since $A_{n+m} \in E_i$, $E_{i_{n+m}} \rightarrow A_{n+m} \in F^+[E_i] \subseteq F^+[J']$. As the last step, we add $E_{i_{n+m}} \rightarrow A_{n+m}$ to F'. Thus, $X_{n+m} \rightarrow A_{n+m}$ is not essential for adding A_{n+m} to $W^+ \cap \overline{J'}$ and hence can be excluded.

We now have excluded one FD in $F - F'$ that can contribute to $W^+ \cap \overline{J'}$. Execute this procedure repeatedly will exclude all FDs in $F - F'$ that can contribute to $W^+ \cap \overline{J'}$. Eventually this procedure will halt since F is finite. Thus, the proof is complete. □

Lemma 4 Let J be a join tree for a reduced, acyclic hypergraph H and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF. Let E_i and E_j be two distinct nodes in J such that $E_i \rightarrow E_j$. Let P be the unique path between E_i and E_j in J. There exists a node E_k on P such that $E_k \neq E_j$, E_k contains a key of E_j as a subset, and $E_i \rightarrow E_k$.

Proof. Figure 12 shows the path P, in which we designate E_a as the neighboring node of E_j. Let P' be the subpath of P from E_i to E_a, including E_i and E_a. If $E_j \subseteq \overline{P'}$, then by Lemma 1, $E_j \subseteq E_a$. This means H is not reduced—a contradiction. Hence, $E_j \not\subseteq \overline{P'}$. Since P is a connected subtree of J, by Lemma 3, $F^+[P]$ implies the FD $E_i \rightarrow E_j$. Thus, $F^+[P]$ implies $E_i \rightarrow K$ for every key K of E_j. If $\overline{P'}$ does not contain any key of E_j as a subset, $\overline{P'}^+ = \overline{P'}$ where $\overline{P'}^+$ is the closure of $\overline{P'}$ under $F^+[P]$. Since $E_i \subseteq \overline{P'}$, $E_i^+ \subseteq \overline{P'}^+$. However, $E_j \not\subseteq \overline{P'} (= \overline{P'}^+)$ implies $E_j \not\subseteq E_i^+$—a contradiction. Thus, $\overline{P'}$ contains a key $\hat{K}$ of E_j as a subset. By Lemma 1, $\hat{K} \subseteq E_a$. Therefore, there exists a node E_b on P such that each of the nodes in between of E_a and E_b on P, including E_a and E_b, contains $\hat{K}$ as a subset; and every node to the left of E_b in Figure 12, if there is any, does not contain any key of E_j. We are left to show $E_i \rightarrow E_b$. If E_i and E_b are the same node, we are done. Assume $E_i \neq E_b$. This implies there is an attribute $A \in (\hat{K} - E_i)$ that does not appear in any node to the left of E_b on P. Let P'' be the subpath of P from E_i to E_b, including E_i and E_b. Since $E_i \rightarrow K$ for every key K of E_j, $E_i \rightarrow \hat{K}$. Since P'' is a connected subtree of J, by Lemma 3, $F^+[P'']$ implies the FD $E_i \rightarrow \hat{K}$. Because A does not appear in any node to the left of E_b, it follows that $E_i \rightarrow K$ where $K \rightarrow A$ is a nontrivial FD in $F^+[E_b]$. Since E_b is in BCNF, $K \rightarrow E_b$ and thus $E_i \rightarrow E_b$. □

[6]That is, $S \cup (X_k \cap \overline{J'})$ is E_{i_k}.

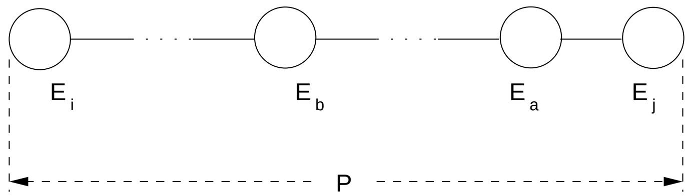


Figure 12: The Path P between E_i and E_j of Lemma 4.

Example 13 Consider the nodes IV_{15} and BCV_{10} in Figure 4(b). The FDs in Figure 4(a) imply the FD $IV_{15} \rightarrow BCV_{10}$. In the path between IV_{15} and BCV_{10}, the node $BEFIV_{12}$ contains B, a key of BCV_{10}. Also, $IV_{15} \rightarrow BEFIV_{12}$. Therefore, $BEFIV_{12}$ fits the statement of Lemma 4. □

Lemma 5 Let J be a join tree for a reduced, acyclic hypergraph H and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF. Let P be the unique path between two distinct nodes E_i and E_j in J where $E_i \rightarrow E_j$ and for any other node E_k on P such that $E_k \neq E_i$ and $E_k \neq E_j$, $E_i \not\rightarrow E_k$. If E_j is not already a neighboring node of E_i, we can rearrange the nodes on P so that E_j becomes a neighboring node of E_i.

Proof. If E_j is already a neighboring node of E_i, then we are done. Therefore, let us assume E_j is not a neighboring node of E_i. Like Lemma 4, Figure 12 shows the path P between E_i and E_j where E_a is the designated neighboring node of E_j on P. As indicated in Figure 13, we show that the edge $\{E_a, E_j\}$ can be removed, and we can add an edge between E_i and E_j. By so doing, we obtain another join tree for H. We now begin our argument. Since $E_i \not\rightarrow E_k$ for any other node E_k on P where $E_k \neq E_i$ and $E_k \neq E_j$, by Lemma 4, E_i contains a key K of E_j as a subset. Let L be the label of the edge $\{E_a, E_j\}$. By Lemma 1, $K \subseteq L$. If $K \subset L$, then since E_a is in BCNF, $K \rightarrow E_a$. This means $E_i \rightarrow E_a$—a contradiction. Thus, $L = K$. In addition, if E_i contains an attribute $A \in (E_j - K)$, then by Lemma 1, $A \in E_a$—a contradiction. Thus, we conclude that $E_a \cap E_j = E_i \cap E_j = K$. As such, we can remove the edge $\{E_a, E_j\}$ and add an edge between E_i and E_j, as Figure 13 shows. □

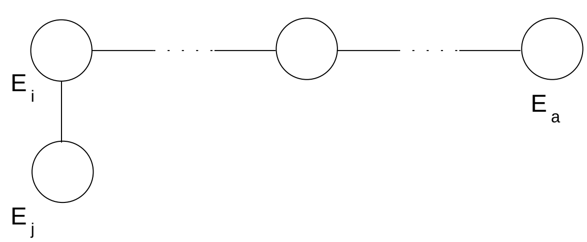


Figure 13: The Rearranged Path of Lemma 5.

Example 14 Consider the nodes ABV_1 and $BHKV_8$ in Figure 4(b). These two nodes fit the statement of Lemma 5, which means we can remove the edge $\{ABV_1, BDV_5\}$ and establish another edge between ABV_1 and $BHKV_8$ with exactly the same label B. □

Theorem 1 Let H be a reduced, acyclic hypergraph and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF. Let C be a set of hyperedges in H such that for any E_i and E_j in C, $E_i \to E_j$ and $E_j \to E_i$ under F. The hypergraph $(H - C) \cup \{\overline{C}\}$ is equivalent to H, and is also acyclic, and each of its hyperedges is in BCNF as well.

Proof. We first consider a simple case, which will be used later in the proof. Suppose J is a join tree for H, and $\{E_i, E_j\}$ is an edge in J such that $E_i \to E_j$ and $E_j \to E_i$ under F. If we create a new node $E_i \cup E_j$ and add it to J, and remove E_i and E_j from J, and at the same time make every edge that was incident with E_i or E_j to be incident with this new node, we obtain a join tree for the hypergraph $H' = (H - \{E_i, E_j\}) \cup \{E_i \cup E_j\}$. Hence, H' is acyclic. To show that H' is equivalent to H, observe that because $\{E_i, E_j\}$ is an edge in J, $E_i \to E_j$ and $E_j \to E_i$, then by Lemma 4, E_i includes a key of E_j and E_j includes a key of E_i. As such, every key of E_i implies the key of E_j that is included in E_i. Likewise, every key of E_j implies the key of E_i that is included in E_j. Therefore, every key of E_i is equivalent to every key of E_j. This means H and H' are equivalent. Now suppose $X \to A$ is a nontrivial FD that holds in $E_i \cup E_j$. By Lemma 3, $X \to A$ is implied by $F^+[E_i] \cup F^+[E_j]$. Since both E_i and E_j are in BCNF, if X does not include any key of E_i or E_j, $X = X^+$ and $F^+[E_i] \cup F^+[E_j]$ does not imply $X \to A$—a contradiction. Therefore, X includes at least one key of E_i or E_j. Since every key of E_i is equivalent to every key of E_j, then $X \to E_i$ and $X \to E_j$, which implies $X \to (E_i \cup E_j)$. Thus, $E_i \cup E_j$ is in BCNF. By repeatedly applying the procedure specified in the proof for Lemma 5 and merging two functionally equivalent nodes that are neighbors, as in the case we just discussed, we can reduce the number of pairs of functionally equivalent nodes to zero. The proof is then complete. □

5.2 NNF and Connected Subtrees

Theorem 2, the main result of this section, states that if we want to construct an NNF scheme tree that syntactically covers some hyperedges, the hyperedges must be the nodes in a connected subtree of a join tree. Otherwise, there will be a violation of NNF.

Theorem 2 Let H be a reduced, acyclic hypergraph and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF and no two distinct hyperedges in H are functionally equivalent. Let T be an NNF scheme tree that is a syntactic cover of a set S of hyperedges in H. The hyperedges in S are precisely the nodes of a connected subtree of a join tree for H (i.e., there exists a join tree J for H such that for any two hyperedges E_p and E_q in S, the path between E_p and E_q in J only includes S's hyperedges).

Proof. Let us assume that S's hyperedges are not the nodes of a connected subtree of any join tree for H. This assumption implies that S contains two distinct hyperedges E_p and E_q in H such that the path between E_p and E_q in any join tree for H includes some hyperedges in $H - S$. Let J be a join tree for H such that the path P between E_p and E_q in J is the shortest among all the possible paths between E_p and E_q. Figure 14 shows the path P and a subpath P' of P, where the endpoints of P', namely E_i and E_j, are the only hyperedges on P' that are in S. Since E_i and E_j are the only nodes on P' that are in S, removing $E_i \cap E_j$ from S will generate at least two connected components C_i and C_j where $(E_i - E_j) \subseteq C_i$ and $(E_j - E_i) \subseteq C_j$. That is, C_i and C_j are two connected components of the hypergraph $\{E - (E_i \cap E_j) : E \text{ is a hyperedge of } S\} - \{\emptyset\}$. This means S generates the nontrivial MVDs $(E_i \cap E_j) \twoheadrightarrow C_i$ and $(E_i \cap E_j) \twoheadrightarrow C_j$. Since T is an NNF scheme tree that syntactically covers S and S generates these two MVDs on $Aset(T)$, by NNF's Condition 1, $MVD(T) \cup FD(T)$ implies both of these MVDs on $Aset(T)$. Nevertheless, if $H \cup F$ does not imply neither of them, then T violates NNF's Condition 1 because $MVD(T) \cup FD(T)$ implies some MVDs on $Aset(T)$ that do not follow from $H \cup F$. This will give us a contradiction, which means our assumption is wrong.

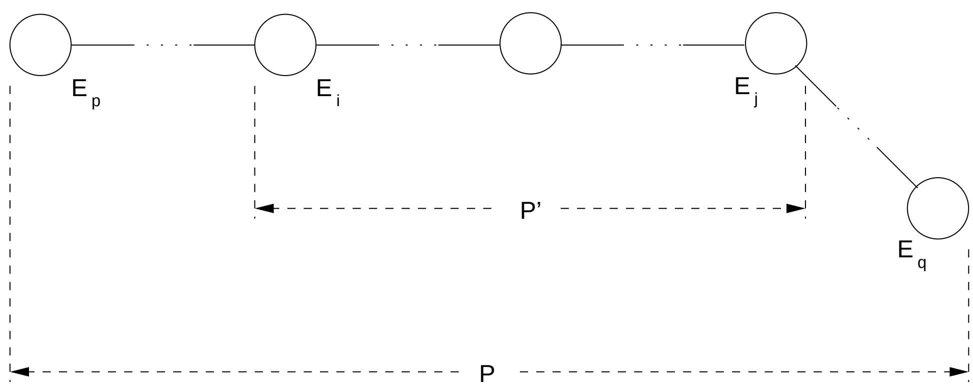


Figure 14: The Subpath P' of Theorem 2.

To show $H \cup F$ does not imply neither $(E_i \cap E_j) \twoheadrightarrow C_i$ nor $(E_i \cap E_j) \twoheadrightarrow C_j$, we first establish several claims. First, we claim that $E_i \cap E_j$ is a proper subset of every label on the path P' in Figure 14. Assume not; we derive a contradiction as follows. By Lemma 1, $E_i \cap E_j$ is a subset of every label on P'. Let $\{E'_i, E'_j\}$ be an edge on P' such that its label is equal to $E_i \cap E_j$ (i.e., $E'_i \cap E'_j$ $= E_i \cap E_j$), and as Figure 15 shows, E'_i and E'_j are chosen in such a way that E'_i is closer to E_i and E'_j is closer to E_j. By our assumption, P' includes at least one hyperedge in $H - S$ as a node. Then, $E_i \neq E'_i$ or $E'_j \neq E_j$. Since $(E'_i \cap E'_j) \subseteq E_i$, if $E_i \neq E'_i$, then we can remove the edge $\{E'_i$, $E'_j\}$ from J and add an edge between E_i and E'_j to obtain another join tree for H with a shorter path between E_i and E_j, and thus a shorter path for E_p and E_q—a contradiction. We will obtain a similar contradiction if $E'_j \neq E_j$. Therefore, $E_i \cap E_j$ is a proper subset of every label on P'. Our second claim is that $(E_i \cap E_j) \not\rightarrow A$ for any $A \in (\overline{P'} - (E_i \cap E_j))$. Assume not, then let E be a node on P' that contains an attribute A such that $(E_i \cap E_j) \rightarrow A$ is nontrivial. Let E' be a neighboring node of E on P'. By our first claim, $(E_i \cap E_j) \subset (E \cap E')$. Therefore, $A \in E$ and $(E_i \cap E_j) \subset E$. Since E is in BCNF and $(E_i \cap E_j) \rightarrow A$ is nontrivial in $F^+[E]$, $(E_i \cap E_j) \rightarrow E$. Similarly, because E' is in BCNF and $(E_i \cap E_j) \subset (E \cap E')$, $(E_i \cap E_j) \rightarrow E'$. Thus, E and E' share a key of $E_i \cap E_j$ as a common key and therefore E and E' are functionally equivalent—a contradiction.

Now, consider removing $(E_i \cap E_j)^+$, the closure of $E_i \cap E_j$ under F, from H. By our second claim,

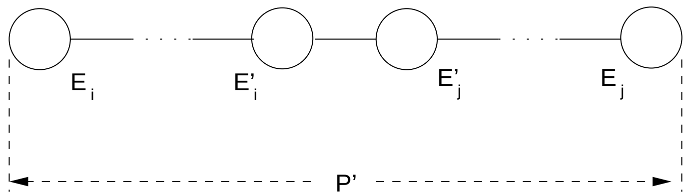


Figure 15: The Edge $\{E_i', E_j'\}$ of Theorem 2.

removing $(E_i \cap E_j)^+$ from the nodes on P' does not remove any more attributes than removing $E_i \cap E_j$ from the nodes on P'. By our first claim, all the nodes on P' remain connected after removing $E_i \cap E_j$ from the nodes on P'. Thus, $E_i - E_j$ and $E_j - E_i$ are both contained as subsets in the same connected component of the hypergraph $\{E - (E_i \cap E_j)^+ : E \text{ is a hyperedge of } H\} - \{\emptyset\}$. Therefore, $H \cup F$ implies neither $(E_i \cap E_j) \twoheadrightarrow C_i$ nor $(E_i \cap E_j) \twoheadrightarrow C_j$ on $Aset(T)$ and the proof is complete. $\Box$

Example 15 A scheme tree with B as the root node and DV_5 and HKV_8 as B's child nodes is not in NNF because BDV_5—$BHKV_8$ is not a connected subtree in Figure 4(b). Removing B from the join tree in Figure 4(b) will not separate DV_5 and HKV_8, as the scheme tree implies. $\Box$

5.3 NNF and Critical Nodes

Theorem 3, the main result of this section, ties critical nodes and connected subtrees together. It states that there exists an NNF scheme tree that syntactically covers the nodes in a connected subtree S of a join tree if and only if S does not have a critical node with respect to S.

Lemma 6 All join trees for a reduced, acyclic hypergraph have the same multiset of labels.
Proof. Let H be an acyclic hypergraph. If H has only one hyperedge, the join tree for H has a single node and no label. The empty set of labels is vacuously unique. Assume this lemma is true if H has k ($k \geq 1$) or less hyperedges. Consider the case that H has $k+1$ hyperedges. Since H is acyclic, H has a join tree J. Arbitrarily choose a leaf node E_L in J. Since E_L is a leaf node, removing E_L from J results in a join tree for the acyclic hypergraph $H - \{E_L\}$. Since $H - \{E_L\}$ is acyclic and has k hyperedges, by the induction hypothesis, $H - \{E_L\}$ has a unique multiset of labels. Consider the edge that connects E_L to another node in J. That edge has the label $E_L \cap (\cup_{E \in (H - \{E_L\})} E)$, which is determined only by E_L and $H - \{E_L\}$ and not by J. Thus, reattaching E_L back to J gives us a unique multiset of labels for H. $\Box$

Lemma 7 Let H be a reduced, acyclic hypergraph and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF and no two distinct hyperedges in H are functionally equivalent. Let $\{L_1, \ldots, L_n\}$ be an equivalence class (a set) of $n \geq 1$ functionally equivalent labels of H. For any i and j such that $1 \leq i, j \leq n$ and $i \neq j$, $L_i \to L_j$ is nontrivial.
Proof. Assume not; we derive a contradiction as follows. First, observe that since $L_1, \ldots, L_n$ are

labels in the same set (not multiset), $L_1, \ldots, L_n$ are all distinct. So, if there are i and j such that $1 \le i, j \le n$, $i \ne j$ and $L_i \to L_j$ is trivial, $L_j \subset L_i$. Since L_i is a label, there is an edge $\{E_{i_1}, E_{i_2}\}$ in a join tree for H such that E_{i_1} and E_{i_2} are hyperedges in H, and $E_{i_1} \cap E_{i_2} = L_i$. Since L_i and L_j are functionally equivalent, $L_j \to L_i$. Since $L_j \subset L_i$, $L_j \to L_i$, and E_{i_1} and E_{i_2} are both in BCNF, E_{i_1} and E_{i_2} share a key of L_j as a common key. This implies E_{i_1} and E_{i_2} are functionally equivalent—a contradiction. □

Lemma 8 Let H be a reduced, acyclic hypergraph and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF and no two distinct hyperedges in H are functionally equivalent. For any two distinct and functionally equivalent labels L_i and L_j of H, there is a unique hyperedge $E \in H$ such that $(L_i \cup L_j) \subseteq E$. Further, L_i and L_j are keys of E and there is a connected subtree like the one in Figure 16(b) in any join tree for H.

Proof. Suppose that H has no hyperedge that includes $L_i \cup L_j$ as a subset. Let J be a join tree for H. Since L_i and L_j are labels of H, there are two nodes E_i and E_j in J such that $L_i \subseteq E_i$ and $L_j \subseteq E_j$. By our assumption, $L_j \not\subseteq E_i$ and $L_i \not\subseteq E_j$. Without loss of generality, E_i and E_j are chosen in such a way that the path P between them in J is the shortest among all possible paths in J. As such, except E_j, no node on P includes L_j as a subset. Similarly, except E_i, no node on P includes L_i as a subset. Let E_a be the neighboring node of E_j on P, as Figure 16(a) shows. (It is possible that E_a and E_i are the same node.) Since $L_j \not\subseteq E_a$, there is an attribute $A \in L_j$ such that $A \notin E_a$. Additionally, A is not in any node to the left of E_a in Figure 16(a); otherwise by Lemma 1, $A \in E_a$—a contradiction.

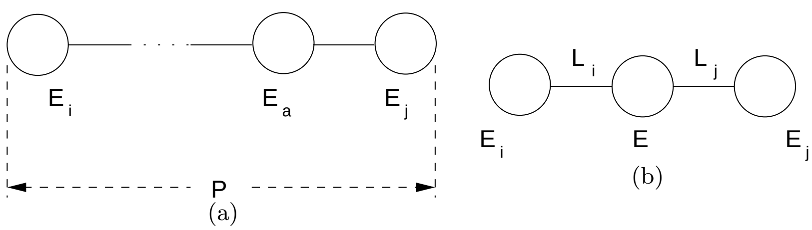


Figure 16: The Path P between E_i and E_j and the Connected Subtree of Lemmas 8 and 9.

Since P is a connected subtree of J, by Lemma 3, $F^+[P]$ implies the FD $L_i \to L_j$. With respect to Figure 16(a), let P' be the subpath of P from E_i to E_a, including E_i and E_a. Since $F^+[P]$ implies $L_i \to L_j$ and $A \in (E_j - \overline{P'})$, it must be that $F^+[P]$ implies $L_i \to K$ where $K \to A$ is a nontrivial FD in $F^+[E_j]$. Since E_j is in BCNF, $K \to E_j$. This implies $L_i \to E_j$, which results in $E_i \to E_j$ because $L_i \subseteq E_i$. Similarly, we can show that $E_j \to E_i$. Thus, E_i and E_j are functionally equivalent—a contradiction.

To show that there is only one hyperedge $E \in H$ that includes $L_i \cup L_j$ as a subset, assume there are two distinct hyperedges E_i and E_j such that $(L_i \cup L_j) \subseteq E_i$ and $(L_i \cup L_j) \subseteq E_j$. Since $L_i \to L_j$ is nontrivial and E_i and E_j are both in BCNF, $L_i \to E_i$ and $L_i \to E_j$. This means E_i and E_j share a key of L_i as a common key. Thus, E_i and E_j are two functionally equivalent hyperedges in H—a contradiction.

We now show that there is a connected subtree like the one in Figure 16(b) in any join tree for H. Observe that since L_i and L_j are labels of H, there are two other hyperedges E_p and E_q in H such that E_p includes L_i but not L_j as a subset and E_q includes L_j but not L_i as a subset. Let E_i be the neighboring node of E on the path between E_p and E. By Lemma 1, $L_i \subseteq (E_i \cap E)$. Since $L_i \to L_j$ is nontrivial, $(L_i \cup L_j) \subseteq E$, and E is in BCNF, $L_i \to E$. Assume $L_i \subset (E_i \cap E)$. Since E_i is also in BCNF and $L_i \to E$, $L_i \to E_i$ as well. This means E_i and E are functionally equivalent—a contradiction. Therefore, the label of the edge $\{E_i, E\}$ is L_i, as Figure 16(b) shows. In addition, if there exists a proper subset L_i' of L_i such that $L_i' \to L_i$, we will reach the same contradiction because $L_i \to E$. Therefore, L_i is a key of E. The same results can be similarly established for L_j and thus the proof is now complete. $\Box$

Lemma 9 Let H be a reduced, acyclic hypergraph and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF and no two distinct hyperedges in H are functionally equivalent. Let C_i and C_j be two distinct equivalence classes of labels of H such that C_j is a parent node of C_i in the Hasse diagram of the partial order $\succeq$ of H. Suppose that for each $L_i \in C_i$ and for each $L_j \in C_j$, $L_j \nsubseteq L_i$. There exists a pair of labels $(L_i, L_j) \in C_i \times C_j$ and a unique hyperedge $E \in H$ such that $(L_i \cup L_j) \subseteq E$. Further, L_i is a key of E and there is a connected subtree like the one in Figure 16(b) in any join tree for H.

Proof. Let J be a join tree for H. Assume there is no node in J that includes $L_i \cup L_j$ as a subset for every pair of labels $(L_i, L_j) \in C_i \times C_j$. Choose two nodes E_i and E_j in J such that the path P between E_i and E_j is the shortest under the requirements that $L_i \subseteq E_i$, $L_j \subseteq E_j$, and $(L_i, L_j) \in C_i \times C_j$. By our assumption, except E_j, no node on P includes L_j as a subset. Similarly, except E_i, no node on P includes L_i as a subset. Let E_a be the neighboring node of E_j on P, as Figure 16(a) shows. (It is possible that E_a and E_i are the same node.) Since $L_j \nsubseteq E_a$, there is an attribute $A \in L_j$ such that $A \notin E_a$. Additionally, A is not in any node to the left of E_a in Figure 16(a); otherwise by Lemma 1, $A \in E_a$—a contradiction.

Since P is a connected subtree of J, by Lemma 3, $F^+[P]$ implies the FD $L_i \to L_j$. With respect to Figure 16(a), let P' be the subpath of P from E_i to E_a, including E_i and E_a. Since $F^+[P]$ implies $L_i \to L_j$ and $A \in (E_j - \overline{P'})$, it must be that $F^+[P]$ implies $L_i \to K$ where $K \to A$ is a nontrivial FD in $F^+[E_j]$. Since E_j is in BCNF, $K \to E_j$. This implies $L_i \to E_j$, which means $L_i \to K$ for any key K of E_j. If $\overline{P'}$ does not include a key of E_j as a subset, then $\overline{P'}^+ = \overline{P'}$ under $F^+[P]$. However, since $L_i \subseteq \overline{P'}$ and $L_j \nsubseteq \overline{P'}$, then $F^+[P]$ does not imply $L_i \to L_j$—a contradiction. Therefore, $\overline{P'}$ includes a key $\hat{K}$ of E_j as a subset. Thus, by Lemma 1, $\hat{K} \subseteq (E_a \cap E_j)$. If $\hat{K} \subset (E_a \cap E_j)$, then because E_a is in BCNF, E_a and E_j share $\hat{K}$ as a common key, which means E_a and E_j are functionally equivalent—a contradiction. Thus, $\hat{K} = E_a \cap E_j$. Observe that $L_j \not\to \hat{K}$; otherwise, $\hat{K} \in C_j$ and therefore E_i and E_a should have been chosen in the first place—a contradiction. Thus, $L_i \to \hat{K}$ where $\hat{K}$ is the label of the edge $\{E_a, E_j\}$ in Figure 16(a). In turn, $\hat{K} \to L_j$ and $L_j \not\to \hat{K}$. This means C_j is not a parent node of C_i in the Hasse diagram of the partial order $\succeq$ of H—a contradiction. Therefore, there is a pair of labels $(L_i, L_j) \in C_i \times C_j$ and a hyperedge $E \in H$ such that $(L_i \cup L_j) \subseteq E$.

To show that E is unique, we may reuse the third paragraph of the proof for Lemma 8. To show that L_i is a key of E and there is a connected subtree like the one in Figure 16(b) in any join tree for H, we may reuse the fourth paragraph of the proof for Lemma 8 for the label L_i. For the label L_j, observe that if the label of the edge $\{E, E_j\}$ is not L_j, then it must be a proper superset of L_j. Since E and E_j are both in BCNF, if $L_j \rightarrow (E \cap E_j)$, then E and E_j are functionally equivalent—a contradiction. Therefore, $L_j \not\rightarrow (E \cap E_j)$. Since L_i is a key of E, $L_i \rightarrow (E \cap E_j)$. This implies C_j is not a parent node of C_i in the Hasse diagram of the partial order $\succeq$ of H—a contradiction. Thus, the label of the edge $\{E, E_j\}$ is L_j. The proof is now complete. □

Example 16 Consider the labels J and B in Figure 4(b). The FD $J \rightarrow B$ is nontrivial. By Lemma 9, there is a unique node that contains JB as a subset, which is $BFJV_{11}$ in Figure 4(b). □

Lemma 10 Let H be a reduced, acyclic hypergraph and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF and no two distinct hyperedges in H are functionally equivalent. Let J be a join tree for H and S be a connected subtree of J. If there is a node in S that is critical with respect to S, then there does not exist an NNF scheme tree that syntactically covers the set of nodes in S.

Proof. Suppose T is an NNF scheme tree that syntactically covers the set of nodes in S. Let E be such a critical node in S and L_i and L_j be two labels belonged to S such that $L_i \not\rightarrow L_j$, $L_j \not\rightarrow L_i$, and $(L_i \cup L_j) \subseteq E$. Since $L_i \not\rightarrow L_j$ and $L_j \not\rightarrow L_i$, E must be on the path between L_i and L_j, as Figure 17 shows. With respect to Figure 17, $L_i \twoheadrightarrow C_i$ is a hypergraph-generated MVD where $C_i \supseteq (E_i - L_i) \neq \emptyset$ and C_i is a connected component of the hypergraph $\{E - L_i : E \text{ is a hyperedge of } S\} - \{\emptyset\}$. Similarly, $L_j \twoheadrightarrow C_j$ is a hypergraph-generated MVD where $C_j \supseteq (E_j - L_j) \neq \emptyset$ and C_j is a connected component of the hypergraph $\{E - L_j : E \text{ is a hyperedge of } S\} - \{\emptyset\}$.

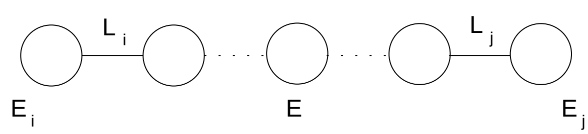


Figure 17: The Labels L_i, L_j, and the Critical Node E of Lemma 10.

Since T syntactically covers the set of nodes in S, $L_i \twoheadrightarrow C_i$ holds for T. Therefore, we need to test it against NNF's Condition 1, which stipulates that $MVD(T)$ and $FD(T)$ must imply $L_i \twoheadrightarrow C_i$. By Lemma 4.5 in [19] and Lemma 3 of this paper, $MVD(T)$ and $FD(T)$ imply $L_i \twoheadrightarrow C_i$ if and only if $MVD(T)$ implies $L_i^+ \twoheadrightarrow C_i$ where L_i^+ is the closure of L_i under $F^+[S]$. Proposition 4.1 in [21] states that $MVD(T)$ is equivalent to the join dependency (JD) $\bowtie\{\overline{P_1}, \ldots, \overline{P_n}\}$ where $\overline{P_k}$ denotes the union of the nodes in the path P_k of T and $P_1, \ldots, P_n$ are all the paths in T. In addition, $L_i^+ \twoheadrightarrow C_i$ is equivalent to the JD $\bowtie\{L_i^+C_i, L_i^+(\overline{S} - L_i^+C_i)\}$. (Note that $\overline{S} = Aset(T)$, the set of attributes of T.) Therefore, $MVD(T)$ implies $L_i^+ \twoheadrightarrow C_i$ if and only if for every path P_k in T, $\overline{P_k}$

$\subseteq L_i^+C_i$ or $\overline{P_k} \subseteq L_i^+(\overline{S} - L_i^+C_i)$ (see Chapter 8 in [16]). Likewise, $MVD(T)$ implies $L_j^+ \twoheadrightarrow C_j$ if and only if for every path P_k in T, $\overline{P_k} \subseteq L_j^+C_j$ or $\overline{P_k} \subseteq L_j^+(\overline{S} - L_j^+C_j)$.

We are now ready to derive a contradiction. We assume E_i, E, and E_j each appears in (not necessarily distinct) paths P_i, P, and P_j in T respectively. Since $L_i \not\rightarrow L_j$, there is an attribute $A_j \in L_j$ such that $A_j \notin L_i^+$. Similarly, since $L_j \not\rightarrow L_i$, there is an attribute $A_i \in L_i$ such that $A_i \notin L_j^+$. By Lemma 1, $A_j \notin C_i$; otherwise $A_j \in L_i$—a contradiction. Likewise, $A_i \notin C_j$; otherwise $A_i \in L_j$—a contradiction. Therefore, $A_i \notin L_j^+C_j$ and $A_j \notin L_i^+C_i$. Since $A_i \in L_i$, $A_j \in L_j$ and $(L_i \cup L_j) \subseteq E$, A_i and A_j both appear in P—the path in which E appears. Let N_i and N_j be the (not necessarily distinct) nodes in P that contain A_i and A_j respectively. Since $L_i \not\rightarrow A_j$ and $L_j \not\rightarrow A_i$, $L_i \not\rightarrow N_j$ and $L_j \not\rightarrow N_i$. Now, there are four cases to consider.

(I) $L_i \not\rightarrow E_i, L_j \not\rightarrow E_j$: By NNF's Condition 1, $\overline{P_i} \subseteq L_i^+C_i$ or $\overline{P_i} \subseteq L_i^+(\overline{S} - L_i^+C_i)$. Since $L_i \not\rightarrow E_i$, there is an attribute $A \in E_i$ such that $A \notin L_i^+$. Since $(E_i - L_i) \subseteq C_i$, $A \in C_i$ and thus $A \notin \overline{S} - L_i^+C_i$. Hence, $A \notin L_i^+(\overline{S} - L_i^+C_i)$. Since E_i appears in P_i, $A \in \overline{P_i}$. Thus, it must be that $\overline{P_i} \subseteq L_i^+C_i$. Likewise, $L_j \not\rightarrow E_j$ implies $\overline{P_j} \subseteq L_j^+C_j$. Since E_i and E both contain A_i, P_i and P share N_i as a common node. However, the node N_j must not be a node in P_i; otherwise $A_j \in \overline{P_i}$ and $A_j \notin L_i^+C_i$ imply $\overline{P_i} \not\subseteq L_i^+C_i$—a contradiction. Hence, $N_j \neq N_i$ and N_j must be lower than N_i in P; otherwise N_j is a node in P_i—a contradiction. This, however, means that N_i is a node in P_j because P and P_j share N_j as a common node. This implies $A_i \in \overline{P_j}$. Nevertheless, $A_i \in \overline{P_j}$ and $A_i \notin L_j^+C_j$ imply $\overline{P_j} \not\subseteq L_j^+C_j$—a contradiction.

(II) $L_i \not\rightarrow E_i, L_j \rightarrow E_j$: Since $L_j \rightarrow E_j$, L_j is a key of E_j; otherwise E_j and its neighboring node in Figure 17 are functionally equivalent—a contradiction. Thus, with respect to Figure 17, for every $A \in (E_j - L_j)$, A does not appear in any node to the left of E_j; otherwise, by Lemma 1, $A \in L_j$, which means L_j is not a key of E_j—a contradiction. Therefore, $A \notin C_i$ for any $A \in (E_j - L_j)$ and if $L_i \rightarrow A$ for some $A \in (E_j - L_j)$, it must be that $L_i \rightarrow K$ where $K \rightarrow A$ is nontrivial in $F^+[E_j]$. Because E_j is in BCNF, $K \rightarrow E_j$. This implies $L_i \rightarrow E_j$, which means $L_i \rightarrow L_j$—a contradiction. Hence, for any $A \in (E_j - L_j)$, $A \notin L_i^+C_i$.

Like in the previous case, $L_i \not\rightarrow E_i$ implies N_i is a node in P_j. For any $A \in (E_j - L_j)$, let N be the node in P_j that contains A. Like A_j, because $A \notin L_i^+C_i$ for any $A \in (E_j - L_j)$ and N_i is a node in P_j, $N \neq N_i$ and N must be lower than N_i in P_j. Thus, for each $A \in (E_j - L_j)$, $L_j \not\rightarrow Ancestor(N)$ because $N_i \subseteq Ancestor(N)$ and $L_j \not\rightarrow N_i$. Therefore, since $L_j \rightarrow A$ nontrivially for each $A \in (E_j - L_j)$, T violates NNF's Condition 2—a contradiction.

(III) $L_i \rightarrow E_i, L_j \not\rightarrow E_j$: This case is symmetrical to the previous case.

(IV) $L_i \rightarrow E_i, L_j \rightarrow E_j$: As we have already proved, $L_j \rightarrow E_j$ implies there is an attribute $A'_j \in (E_j - L_j)$ such that $L_i \not\rightarrow A'_j$. Likewise, $L_i \rightarrow E_i$ implies there is an attribute $A'_i \in (E_i - L_i)$ such that $L_j \not\rightarrow A'_i$. Let N'_i and N'_j be the nodes in T that contain A'_i and A'_j respectively. Since $L_i \not\rightarrow A'_j$ and $L_j \not\rightarrow A'_i$, $L_i \not\rightarrow N'_j$ and $L_j \not\rightarrow N'_i$. Without loss of generality, we assume $N_j = N_i$ or N_j is higher than N_i in P. As such, N_j is on both P_i and P_j. Since N_j is on P_i and $L_i \rightarrow A'_i$ nontrivially, N'_i must be higher than N_j in P_i because $L_i \not\rightarrow N_j$; otherwise T violates NNF's Condition 2—a contradiction. This implies N'_i is on both P_i and P_j. Further, since N'_i is on P_j

and $L_j \rightarrow A'_j$ nontrivially, N'_j must be higher than N'_i in P_j because $L_j \not\rightarrow N'_i$. This also means N'_j is on both P_i and P_j. Thus, N_i, N_j, N'_i, N'_j, in this order, are all on the same path. However, this will make $N'_j \subseteq Ancestor(N'_i)$. We now have a violation of NNF's Condition 2 because $L_i \rightarrow A'_i$ nontrivially and $L_i \not\rightarrow Ancestor(N'_i)$—a contradiction. $\square$

Lemma 11 Let H be a reduced, acyclic hypergraph and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF and no two distinct hyperedges in H are functionally equivalent. Let J be a join tree for H and S be a connected subtree of J. If there is not a node in S that is critical with respect to S, then the Hasse diagram of the partial order $\succeq$ on S's labels is a rooted tree.

Proof. For each pair of edges $\{E_i, E\}$ and $\{E, E_j\}$ in S, either $(E_i \cap E) \rightarrow (E \cap E_j)$ or $(E \cap E_j) \rightarrow (E_i \cap E)$; otherwise E, a node in S, is critical with respect to S—a contradiction. Therefore, if the Hasse diagram is not a rooted tree, it must have a "V-shape." For example, there are two V-shapes in Figure 5(a). We now show that a V-shape in the Hasse diagram implies it has a critical node with respect to S. By this, we obtain a contradiction. Assume such a V-shape is made up by three equivalence classes C_i, C_j and C_k of functionally equivalent labels in S such that C_i and C_j are two parent nodes of C_k in the Hasse diagram. We have the following cases to consider.

(I) $\forall L_i \in C_i \forall L_k \in C_k(L_i \not\subseteq L_k), \forall L_j \in C_j \forall L_k \in C_k(L_j \not\subseteq L_k)$: Since S is a connected subtree, S itself is also a join tree. By Lemma 9, there exists a pair of labels $(L_i, L_{k_i}) \in C_i \times C_k$ and a unique node $E_i \in S$ such that $(L_i \cup L_{k_i}) \subseteq E_i$. Further, L_{k_i} is a key of E_i. Likewise, there exists a pair of labels $(L_j, L_{k_j}) \in C_j \times C_k$ and a unique node $E_j \in S$ such that $(L_j \cup L_{k_j}) \subseteq E_j$. Further, L_{k_j} is a key of E_j. If $E_i \neq E_j$, then because L_{k_i} and L_{k_j} are functionally equivalent and L_{k_i} and L_{k_j} are keys of E_i and E_j respectively, E_i and E_j are functionally equivalent—a contradiction. Hence, $E_i = E_j$ and E_i is a critical node.

(II) $\forall L_i \in C_i \forall L_k \in C_k(L_i \not\subseteq L_k), \exists L_j \in C_j \exists L_{k_j} \in C_k(L_j \subset L_{k_j})$: By Lemma 9, there exists a pair of labels $(L_i, L_{k_i}) \in C_i \times C_k$ and a unique node $E_i \in S$ such that $(L_i \cup L_{k_i}) \subseteq E_i$. Further, L_{k_i} is a key of E_i. If $L_{k_j} = L_{k_i}$, then E_i is a critical node. Assume $L_{k_j} \neq L_{k_i}$. By Lemma 8, there is a node E_k of which L_{k_i} and L_{k_j} are keys. If $E_i \neq E_k$, then since L_{k_i} is a key for both of them, E_i and E_k are functionally equivalent—a contradiction. Hence, $E_i = E_k$ and thus $L_j \subset L_{k_j} \subset E_i$. Hence, E_i is a critical node.

(III) $\exists L_i \in C_i \exists L_{k_i} \in C_k(L_i \subset L_{k_i}), \forall L_j \in C_j \forall L_k \in C_k(L_j \not\subseteq L_k)$: This case is symmetrical to the previous case.

(IV) $\exists L_i \in C_i \exists L_{k_i} \in C_k(L_i \subset L_{k_i}), \exists L_j \in C_j \exists L_{k_j} \in C_k(L_j \subset L_{k_j})$: If $L_{k_i} = L_{k_j}$, then either one of the two nodes of an edge whose label is L_{k_i} is a critical node. If $L_{k_i} \neq L_{k_j}$, then by Lemma 8, there is a node E_k of which L_{k_i} and L_{k_j} are keys. Hence, E_k is a critical node. $\square$

Lemma 12 Let H be a reduced, acyclic hypergraph and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF and no two distinct hyperedges in H are functionally equivalent. Let J be a join tree for H and S be a connected subtree of J. If there is not a node in S that is critical with respect to S, then there exists an NNF scheme tree that syntactically covers the hyperedges in S.

Proof. By Lemma 11, the Hasse diagram of the partial order $\succeq$ on S's labels is a rooted tree T. Suppose Step 2 of Procedure `AttachHyperedges` finds two nodes N_i and N_j in different paths of T for a node E in S. Thus, there are labels $L_i \in N_i$ and $L_j \in N_j$ such that $(L_i \cup L_j) \subseteq E$. Since N_i and N_j are in different paths of T, $L_i \not\to L_j$ and $L_j \not\to L_i$. This implies E is critical—a contradiction. Hence, we may run Steps 2, 3 and 4 of Procedure `AttachHyperedges` on T to obtain a scheme tree T'.

We first prove by induction on the number n of nodes in T that every node in S appears in a path of T'. If $n = 0$, then T is empty. This implies S has zero or one node. In the former case, our claim is vacuously true. In the latter case, the only node of S becomes the only node of T'. If $n = 1$, T has a single node. Then, all the labels in that node are merged together to form the root node of T' and each node in S forms a path in T'. Therefore, our claim is also true when $n = 1$. Assume our claim is true if $n \leq k$ for some $k \geq 1$. Run Procedure `MoveLabelsToCenterNodes` on S. Let T_k be an NNF skeleton with k nodes. Consider a child node N_c of a node N_p in the Hasse diagram of $\succeq$ where N_p is already a node in T_k. We obtain NNF skeleton T_{k+1} by adding N_c as a child node to N_p in T_k. If $N_c \succeq N_p$ nontrivially, then by Lemma 9 there are labels $L_p \in N_p$, $L_c \in N_c$, and a unique node E in S such that $(L_p \cup L_c) \subseteq E$. If $N_c \succeq N_p$ trivially, then there are labels $L_p \in N_p$, $L_c \in N_c$ such that $L_p \subset L_c$. Let E' be the node of an edge with the label L_p such that if that edge was removed from S, E' would separate from L_c. Observe that for each $L \in N_c$, $L \not\subseteq E'$. Thus, E' must be attached to N_p. By the induction hypothesis, $L_p \subseteq \mathit{Ancestor}(N_p)$ in T'. By Lemma 1, the intersection of a label in N_p and a label in N_c is a subset of L_p. Therefore, since N_c is a child node of N_p in T, every label in N_c is a subset of $\mathit{Ancestor}(N_c)$ in T'. This means that every node in S that is attached to N_c appears in a path of T'. The induction step is thus complete. Since we do not add any attribute to T that is not in any node in S, $\mathit{Aset}(T') = \overline{S}$. Hence, T' syntactically covers the set of nodes in S.

We are left to prove that T' is in NNF. Since S is a connected subtree of J, S itself is also a join tree. Thus, the set of MVDs generated by S is equivalent to $\bowtie\{E_1, \ldots, E_m\}$ where $E_1, \ldots, E_m$ are the nodes in S [3]. Hence, to prove T' satisfies NNF's Condition 1, we need to show that $\mathit{MVD}(T')$ and $\mathit{FD}(T')$ are equivalent to $\bowtie\{E_1, \ldots, E_m\}$ and $F^+[S]$. We stated earlier that $\mathit{MVD}(T')$ is equivalent to $\bowtie\{\overline{P_1}, \ldots, \overline{P_n}\}$ where $P_1, \ldots, P_n$ are all the paths in T' (see the proof for Lemma 10). Also, observe that $\mathit{FD}(T')$ is equivalent to $F^+[S]$. Thus, one direction of the equivalence is easily established because T' syntactically covers the set of nodes in S. For each path P in T', consider P's leaf node $N_E = \{A \in E : A \text{ does not appear in any label in any node of } T\}$ for some hyperedge $E \in S$. Let N_E's parent node be N. Since E contains a label in N, $E \to \mathit{Ancestor}(N)$ in T'. Therefore, $\overline{P} \subseteq E^+$. By Chapter 8 in [16], T' satisfies NNF's Condition 1.

To prove T' satisfies NNF's Condition 2, observe that by Lemma 3 it is sufficient to only consider $F^+[S]$. Thus, let $X \to A$ be a nontrivial FD in $F^+[E]$ for some node E in S. Since E is in BCNF, $X \to E$. Assume E is attached to a node N in T. It is clear that $E \to \mathit{Ancestor}(N)$ in T' and thus $X \to E \cup \mathit{Ancestor}(N)$ in T'. It follows that T' satisfies NNF's Condition 2 as well. □

Theorem 3 Let H be a reduced, acyclic hypergraph and F be a set of embedded FDs in H such that each hyperedge of H is in BCNF and no two distinct hyperedges in H are functionally equivalent. Let J be a join tree for H and S be a connected subtree of J. There exists an NNF scheme tree that syntactically covers the hyperedges in S if and only if there is not a node in S that is critical with respect to S.

Proof. This theorem follows immediately from Lemmas 10, 11 and 12. $\square$

5.4 Correctness

Theorem 4 Procedure `Main` of Section 3.1 generates a largest NNF scheme tree from its input in polynomial time.

Proof. Let T and J respectively be the input NNF skeleton and the input modified join tree of Procedure `AttachHyperedges`. We first show that the set S defined in Step 1 of Procedure `AttachHyperedges` constitutes a connected subtree of J and it does not have a critical node. By Lemmas 7, 8, and 9, if $N_c \succeq N_p$ nontrivially for two nodes N_p and N_c in T where N_p is the parent of N_c, then the edges whose labels are in N_p and N_c clearly form a connected subtree of J. On the other hand, if $N_c \succeq N_p$ trivially, then there are labels $L_p \in N_p$ and $L_c \in N_c$ such that $L_p \subset L_c$. As such, we may make an edge with the label L_p to be incident with the center node of N_c in S. Thus, the edges whose labels are in N_p and N_c also form a connected subtree of J.

We now proceed to prove that S does not have a critical node. Assume not, let L_i and L_j be two labels in T such that $L_i \not\rightarrow L_j$ and $L_j \not\rightarrow L_i$; and let E be a node in S such that $(L_i \cup L_j) \subseteq E$. As such, E must be on the path between L_i and L_j in S, as Figure 17 shows. If there is at least one label L_k between L_i and L_j on that path such that $L_k \not\rightarrow L_i$ and $L_k \not\rightarrow L_j$, then the existence of E will lead to $L_k \rightarrow L_i$ or $L_k \rightarrow L_j$—a contradiction. Let $L_i \in N_i$ and $L_j \in N_j$ and assume N_i and N_j are nodes in T that have different parents. Then, there is at least one label L_k between L_i and L_j in S such that $L_k \not\rightarrow L_i$ and $L_k \not\rightarrow L_j$—a contradiction. Hence, N_i and N_j are child nodes of the same parent N_k in T. As such, $N_i \not\succeq N_j$, $N_j \not\succeq N_i$, $N_k \not\succeq N_i$, and $N_k \not\succeq N_j$. Since N_k is the parent of N_i and N_j in T, there are nodes E_i and E_j in S such that $(L_{k_i} \cup L_i) \subseteq E_i$ and $(L_{k_j} \cup L_j) \subseteq E_j$ where $L_{k_i}, L_{k_j} \in N_k$, $L_i \in N_i$ and $L_j \in N_j$. We now have the following cases to consider.

(I) $N_i \succeq N_k$ nontrivially and $N_j \succeq N_k$ nontrivially: Assume $E_i = E_j$. By Lemma 9, L_i and L_j are keys of E_i. This implies $N_i \succeq N_j$ and $N_j \succeq N_i$—a contradiction. Hence, $E_i \neq E_j$. As such, there is a label $L_k \in N_k$ that is in between of L_i and L_j in S—a contradiction. Hence, there is no node in S that is critical.

(II) $N_i \succeq N_k$ nontrivially and $N_j \succeq N_k$ trivially: Assume $E_i = E_j$. By Lemma 9, L_i is a key of E_i. This implies $N_i \succeq N_j$—a contradiction. Hence, $E_i \neq E_j$. We may now proceed like in the previous case from this point on.

(III) $N_i \succeq N_k$ trivially and $N_j \succeq N_k$ nontrivially: This case is symmetrical to the previous case.

(IV) $N_i \succeq N_k$ trivially and $N_j \succeq N_k$ trivially: Suppose there is a label $L_k \in N_k$ in between of N_i's center node and N_j's center node. Then, there is a label $L_k \in N_k$ in between of L_i and L_j in S—a contradiction. Hence, there is not a label $L_k \in N_k$ in between of N_i's center node and N_j's center node. However, in this case Procedure `CalculateLabelCnt` at best selects one of N_i and N_j or at worst selects none of N_i and N_j in constructing a largest NNF skeleton. Thus, S has no critical nodes.

To prove that Procedure `Main` generates a largest NNF scheme tree, we show that if we add one more node (hyperedge) in J to S, S will have a critical node. Now, suppose we add one more equivalence class C of labels in the Hasse diagram of $\succeq$ to T. Because of Theorem 2, C must be connected to an equivalence class C_T already in T. Further, C cannot be a child node of C_T in the Hasse diagram of $\succeq$ (i.e., $C \not\succeq C_T$); otherwise, Procedure `CalculateLabelCnt` has already considered C in constructing T. Suppose $C_T \not\succeq C$. If the label of the edge between C's center node and C_T's center node is in C, then C_T's center node is a critical node. If the label of the edge between C's center node and C_T's center node is in C_T, then C's center node is a critical node. Now suppose $C_T \succeq C$. Observe that C_T cannot be a root node in the Hasse diagram of $\succeq$; otherwise $C_T \not\succeq C$. Then, there is a V-shape in T, which means C_T's center node is a critical node. $\square$

5.5 Complexity Analysis

We now prove by a worst-case analysis that Procedure `Main` runs in time polynomial in the size of the input. We first consider the two preparatory procedures: Procedure `MergeHyperedges` and Procedure `CreateJoinTree`. Procedure `MergeHyperedges` uses Algorithm 4.4 on page 66 in [16] in its computation. This algorithm has time complexity $O(p)$, where p is the number of symbols required to represent the given set of FDs. Thus, generating the closure E^+ of one hyperedge E takes $O(p)$ time. For $q \geq 1$ hyperedges, it takes $O(pq)$ time to compute the p closures of the q hyperedges. Let n be the number of symbols required to represent the input acyclic hypergraph and the set of embedded FDs. It is easy to see that p and q are proportional to n. Hence, it takes $O(n^2)$ time to compute the q closures. Now, consider merging two hyperedges when their closures are equal. Given $q > 1$ closures over $r > 1$ distinct attributes, we compute the number of comparisons in the worst case that no pair of closures is equal. First, we use a matrix with q rows and r columns to represent these q closures where $cell(i, j)$—the cell at row i and column j—is equal to 1 if closure C_i has attribute A_j; otherwise, $cell(i, j)$ is equal to 0. Filling up this matrix obviously takes $O(n)$ time. With this matrix, closure C_i is equal to closure C_j if and only if $cell(i, 1) = cell(j, 1)$, $cell(i, 2) = cell(j, 2)$, $\ldots$, and $cell(i, r) = cell(j, r)$. Thus, checking whether $C_i = C_j$ takes r comparisons. Proving closure C_1 is not equal to any other closure therefore takes $(q-1)r$ comparisons. For closure C_2, it similarly takes $(q-2)r$ comparisons. The same reasoning applies to all the other closures. Hence, it takes $(q-1)r+(q-2)r+\cdots+r = q(q-1)r/2$ comparisons to prove that no closure is equal to another closure. Since r is also proportional to n, it takes $O(n^3)$ time to show that no pair of closures is equal. As stated in [24], a straightforward implementation for Procedure `CreateJoinTree` runs in time quadratic in the size of the input acyclic hypergraph. Hence, both Procedure `MergeHyperedges` and Procedure `CreateJoinTree` run in polynomial time with respect to n.

As for Procedure `ConstructHasseDiagramOf`$\succeq$, Procedure `MoveLabelsToCenterNodes`, and Procedure `ExtractLargestNNFSkeleton`, our experiments strongly indicate that these three procedures considered as a whole run in time quadratic in the number of hyperedges. However, since the hyperedges and equivalence classes of labels are generated randomly, this can only be considered as an average-case complexity. For a worst-case analysis of Procedure `ConstructHasseDiagramOf`$\succeq$, let n be the number of symbols required to represent the input acyclic hypergraph and the set of embedded FDs. Observe that the number of labels is one less than the number of nodes (hyperedges) in any join tree. Hence, sorting functionally equivalent labels in a join tree into equivalence classes is similar to merging functionally equivalent hyperedges in an acyclic hypergraph. Further, for two distinct labels L_i and L_j, $L_i^+ \subset L_j^+$, $L_j^+ \subset L_i^+$, and $L_i^+ = L_j^+$ can all be tested successively in the same pass. Thus, sorting labels into equivalence classes and generating the partial order $\succeq$ can be done at the same time. Therefore, Procedure `ConstructHasseDiagramOf`$\succeq$ at most takes $O(n^3)$ time. Procedure `MoveLabelsToCenterNodes` at most reorganizes every edge in a join tree once. Thus, Procedure `MoveLabelsToCenterNodes` runs in time linear in the number of labels in a join tree, which is proportional to n. The time complexity of Procedure `ExtractLargestNNFSkeleton` clearly depends on the time complexity of Procedure `CalculateLabelCnt`, which is recursive. This recursive procedure visits each equivalence class of labels in the Hasse diagram of $\succeq$ once as it calculates its *labelCnt*. Hence, Procedure `CalculateLabelCnt` runs in time linear in the number of equivalence classes of labels, which again is proportional to n. Obviously, each other step of Procedure `ExtractLargestNNFSkeleton` has time complexity $O(n)$. Hence, Procedure `ExtractLargest-NNFSkeleton` has time complexity $O(n)$. For Procedure `AttachHyperedges`, note that a reasonable implementation of a label has two pointers that point at the two nodes (hyperedges) to which it connects. Thus, given an NNF skeleton, finding the set S of Step 1 in Procedure `AttachHyperedges` takes time linear in the number of labels in the NNF skeleton. Attaching them to the NNF skeleton then also takes time linear in the number of labels in the skeleton, which is proportional to n. $\square$

6 Concluding Remarks

In this paper we presented a polynomial-time algorithm to generate a largest redundancy-free XML storage structure from an acyclic hypergraph and a set of embedded FDs where each hyperedge is in BCNF. The algorithm generates a largest NNF scheme tree, which can then be mapped to a redundancy-free XML storage structure. Besides reducing space requirements and overcoming update anomalies, the algorithm also determines a largest set of hyperedges such that no join is needed to navigate from one data item to another within the storage structure. Further, when applied repeatedly on hypergraph edges not already included in generated scheme-trees, the algorithm always yields redundancy-free XML storage structures and often, especially in practical cases, yields the fewest. This, then, also reduces the join cost to navigate from any data item within the application to any other.

It is an open problem to determine whether a polynomial-time algorithm exists to generate a minimum number of scheme trees from an acyclic hypergraph and a set of embedded FDs where each hyperedge is in BCNF. However, since NNF is equivalent to BCNF when only flat relation schemes are allowed [19], BCNF's well-known intractable problems might carry over to this open problem. For example, Theorem 4.22 in [14] states that "The problem of finding a lossless join and nonredundant decomposition of schema R that is in BCNF with respect to a set F of FDs over R, and such that the number of relation schemas in R is less than or equal to some natural number $k \geq 1$ is NP-hard." It suffices to say that at this point more research is needed for this open problem.

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] Reema Al-Kamha. *Conceptual XML for Systems Analysis.* PhD dissertation, Department of Computer Science, Brigham Young University, Provo, Utah, June 2007.

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

[3] 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.

[4] Ronald Bourret. XML and databases. September 2005. http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[5] Ronald Bourret. XML database products. March 2007. http://www.rpbourret.com/xml/XMLDatabaseProds.htm.

[6] Yi Chen, Susan B. Davidson, Carmem S. Hara, and Yifeng Zheng. RRXF: Redundancy reducing XML storage in relations. In *Proceedings of 29th International Conference on Very Large Data Bases*, pages 189–200, Berlin, Germany, September 9-12 2003.

[7] David W. Embley and Wai Yin Mok. Developing XML documents with guaranteed "good" properties. In *Proceedings of the 20th International Conference on Conceptual Modeling*, pages 426–441, Yokohama, Japan, November 27-30 2001.

[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] Thorsten Fiebig, Sven Helmer, Carl-Christian Kanne, Guido Moerkotte, Julia Neumann, Robert Schiele, and Till Westmann. Anatomy of a native XML base management system. *The VLDB Journal*, 11(4):292–314, 2002.

[10] Nathan Goodman, Oded Shmueli, and Y. C. Tay. GYO reductions, canonical connections, tree and cyclic schemas and tree projections. In *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 267–278, Atlanta, Georgia, March 21-23 1983.

[11] Gang Gou and Rada Chirkova. Efficiently querying large XML data repositories: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(10):1381–1403, 2007.

[12] H. V. Jagadish, Shurug Al-Khalifa, Adriane Chapman, Laks V. S. Lakshmanan, Andrew Nierman, Stelios Paparizos, Jignesh M. Patel, Divesh Srivastava, Nuwee Wiwatwattana, Yuqing Wu, and Cong Yu. TIMBER: A native XML database. *The VLDB Journal*, 11(4):274–291, 2002.

[13] Solmaz Kolahi and Leonid Libkin. XML design for relational storage. In *Proceedings of the 16th International Conference on World Wide Web*, pages 1083–1092, Banff, Alberta, Canada, May 8-12 2007.

[14] Mark Levene and George Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer, 1999.

[15] Leonid Libkin. Normalization theory for XML. In *Proceedings of the 5th International XML Database Symposium*, pages 1–13, Vienna, Austria, September 23-24 2007.

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

[17] Wai Yin Mok. A comparative study of various nested normal forms. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):369–385, 2002.

[18] 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.

[19] 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.

[20] Matthias Nicola and Bert Van der Linden. Native XML support in DB2 universal database. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 1164–1174, Trondheim, Norway, August 30 - September 2 2005.

[21] Z. Meral Özsoyoglu and Li-Yan Yuan. A new normal form for nested relations. *ACM Transactions on Database Systems*, 12(1):111–136, 1987.

[22] 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 31st - February 3rd 2005.

[23] Bart Steegmans, Ronald Bourret, Owen Cline, Olivier Guyennet, Shrinivas Kulkarni, Stephen Priestley, Valeriy Sylenko, and Ueli Wahli. *XML for DB2 Information Integration*. IBM, July 2004.

[24] Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[25] 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.

[26] 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 31st - February 3rd 2005.

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