Degrees of Acyclicity for Hypergraphs and Relational Database Schemes

RONALD FAGIN

IBM Research Laboratory, San Jose, California

Abstract. Database schemes (which, intuitively, are collections of table skeletons) can be viewed as hypergraphs (A *hypergraph* is a generalization of an ordinary undirected graph, such that an edge need not contain exactly two nodes, but can instead contain an arbitrary nonzero number of nodes.) A class of "acyclic" database schemes was recently introduced. A number of basic desirable properties of database schemes have been shown to be equivalent to acyclicity This shows the naturalness of the concept. However, unlike the situation for ordinary, undirected graphs, there are several natural, nonequivalent notions of acyclicity for hypergraphs (and hence for database schemes). Various desirable properties of database schemes are considered and it is shown that they fall into several equivalence classes, each completely characterized by the degree of acyclicity of the scheme The results are also of interest from a purely graph-theoretic viewpoint. The original notion of acyclicity has the counterintuitive property that a subhypergraph of an acyclic hypergraph can be cyclic. This strange behavior does not occur for the new degrees of acyclicity that are considered.

Categories and Subject Descriptors: G.2.2 **[Discrete Mathematics]**: Graph Theory—*graph algorithms; trees*; H.2.1 **[Database Management]**: Logical Design—*normal forms; schema and subschema*; H 3.3 **[Information Storage and Retrieval]**: Information Search and Retrieval—*query formulation*

General Terms: Algorithms, Design, Languages, Management, Theory

Additional Key Words and Phrases. Acyclicity, hypergraph, database scheme, relational database, join dependency, loop-free Bachman diagram

1. *Introduction*

A *hypergraph* is a pair $(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a finite set of *nodes* and $\mathcal{E}$ is a set of *edges* (or *hyperedges*) which are arbitrary nonempty subsets of $\mathcal{N}$. An ordinary undirected graph (without self-loops) is, of course, a hypergraph where every edge has exactly two nodes. A special class of hypergraphs, called *acyclic*, has recently been introduced [7, 8, 22, 23]. We shall call this class *α-acyclic* in this paper. There is a natural correspondence between database schemes, each of which can be thought of as a collection of table skeletons, as in Figure 1, and hypergraphs. For example, the hypergraph of Figure 2 corresponds to the database scheme of Figure 1. A database scheme is said to be α-acyclic precisely if the corresponding hypergraph is. Every α-acyclic database scheme enjoys a number of desirable properties, each of which is in fact equivalent to α-acyclicity [7, 8, 22, 23, 25, 32]. Further [38], there are

Most of this research was conducted while the author was a Visiting Research Fellow at Pontificia Universidade Catolica do Rio de Janeiro and was supported in part by a grant from IBM Brazil

Author's address: IBM Research Laboratory K51/281, 5600 Cottle Road, San Jose, CA 95193.

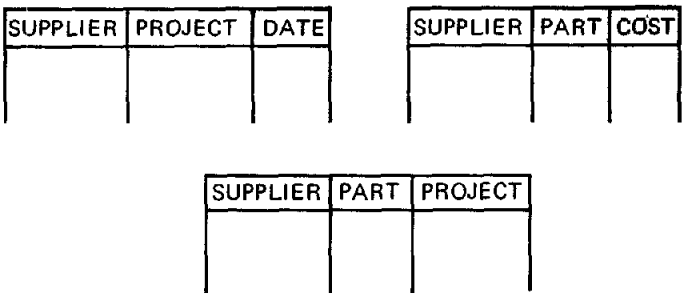


FIGURE 1

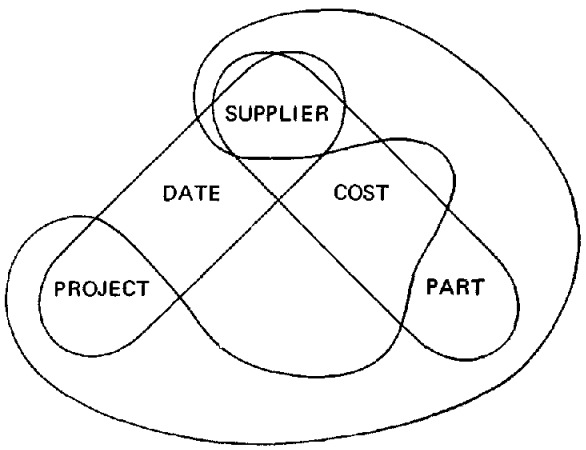


FIGURE 2

problems that are NP-complete in general, but which have polynomial-time algorithms under the assumption of α-acyclicity.

There are other, even nicer properties of database schemes that are too strong to be obeyed by *all* α-acyclic database schemes. We study some such properties and characterize graph-theoretically those database schemes which enjoy these properties. Once again, the properties fall into equivalence classes, which correspond to natural "degrees of acyclicity" for hypergraphs. For, unlike the situation for ordinary, undirected graphs, there are a number of inequivalent, natural definitions of acyclicity for hypergraphs. It is appropriate to speak of "degrees of acyclicity," rather than simply "types of acyclicity," since it turns out that there is a linear ordering of the strengths of the types of acyclicity we consider; the weakest (least restrictive) is the previously studied notion of α-acyclicity.

Our new degrees of acyclicity remedy a mathematically unnatural property of the earlier notion of α-acyclicity; namely, it is possible for a hypergraph to be α-acyclic but have an α-cyclic subhypergraph. (A *subhypergraph* contains a subset of the edges of the original hypergraph.) This strange phenomenon does not occur for our new degrees of acyclicity.

Each of the degrees of hypergraph acyclicity that we consider is a generalization of the concept of acyclicity for ordinary undirected graphs; that is, an undirected graph is acyclic in the usual sense if and only if it is "acyclic," when viewed as a hypergraph, for any of our notions of "acyclic."

There is an analogy between degrees of acyclicity for database schemes and normal forms [15, 20] for relation schemata (a relation scheme along with its set of dependencies [21]). For, there is a hierarchy of normal forms for relation schemata, each

normal form being more restrictive than its predecessor. Codd has argued that we should not *insist* that a relation schema be in a given normal form. Rather, the database designer should be aware of the issues and have a warning flag that if the relation schema is not in a given normal form, then certain problems may arise. An identical comment applies to the question of whether a database scheme should obey a given degree of acyclicity. In practice, it might be reasonable to try to attain a given degree of acyclicity in a user's view (which involves only a portion of the database), rather than in the whole database scheme. This might be attainable, for example, by renaming attributes. An example is given in Section 8.

We now give an example of a natural database property that is equivalent to one of our degrees of acyclicity ("γ-acyclicity"). Assume that there are (among others) an EMP_INFO relation with attributes (column names) EMP (for "employee"), DEPT, and SALARY, and a DEPT_INFO relation with attributes DEPT, CITY, and MGR. An example of an "{EMP, CITY} relationship" is obtained by joining together the EMP_INFO and the DEPT_INFO relations on DEPT and projecting the result on EMP and CITY. It is conceivable that there could be other {EMP, CITY} relationships, obtained by taking one, two, or more other relations and joining them together in some manner and then projecting the result onto EMP and CITY. However, we show that a database scheme is γ-acyclic if and only if for every set X of attributes (such as {EMP, CITY}) and every consistent database over the scheme, there is a unique X-relationship.

Thus, in the above example, if the database scheme is γ-acyclic and the database is consistent, then there is a unique {EMP, CITY} relationship. This fact has a number of useful corollaries. For example, an SQL query [13] to find all EMPs associated with the CITY San Jose would be

```
SELECT EMP
FROM EMP_INFO, DEPT_INFO
WHERE EMP_INFO.DEPT = DEPT_INFO.DEPT
AND DEPT_INFO.CITY = 'San Jose'.
```

However, by γ-acyclicity it is possible instead to unambiguously pose the query

```
SELECT EMP WHERE CITY = 'San Jose'.        (1.1)
```

The desirability of being able to pose queries such as (1.1), with such a simple syntax, has been discussed by Ullman [37]. Not only is the latter query easier to pose and simpler to understand than the SQL query, but also the system has a great deal of flexibility in optimizing how to find the result of the query. The system's choice of which relations to join might depend, for example, on which indices are present.

Languages such as SQL are considered high-level, since it is not necessary to explicitly state the access paths (such as which indices to utilize). Similarly, in a γ-acyclic database scheme it is possible to make use of a still higher level language, in which it is not even necessary to specify which relations must be joined to obtain the answer the user desires.

We now discuss the organization of the paper. If all the reader cares about are the database properties (as opposed to graph-theoretic properties), then he can simply skim Sections 2–6; for example, such a reader need only note one of the various equivalent definitions of a given degree of acyclicity. In Section 2 we present some basic definitions and define α-acyclicity. In Section 3 we define Berge's [10] notion of acyclicity. In Section 4 we give several natural but different-looking definitions of one of our new degrees of acyclicity, namely, β-acyclicity, and prove that the

definitions are equivalent. We also discuss the desirability of β-acyclic database schemes. In Section 5 we define γ-acyclicity and prove the equivalence of various definitions of γ-acyclicity. In Section 6 we prove that Berge-acyclicity $\Rightarrow$ γ-acyclicity $\Rightarrow$ β-acyclicity $\Rightarrow$ α-acyclicity but that none of the reverse implications hold. We also contrast features of the various degrees of acyclicity and discuss their naturalness. In Section 7 we define join expressions, which correspond to "programs" for taking joins, and discuss some of their properties (join expressions are useful for the discussion in Section 8). In Section 8 we describe a number of desirable properties of database schemes, involving monotone-increasing joins and unique relationships among attributes, such that each property is equivalent to the scheme being γ-acyclic. In Section 9 we give polynomial-time algorithms for determining the degree of acyclicity. In Section 10 we present our conclusions.

2. *α-acyclicity*

Let $\mathcal{N}$ be a finite set of distinct symbols, called *attributes* (or *column names*), and let Y be a subset of $\mathcal{N}$. A *Y-tuple* (or simply a *tuple*, if Y is understood) is a function with domain Y. Thus a tuple is a mapping that associates a value with each attribute in Y. If X is a subset of Y and t is a Y-tuple, then $t[X]$ denotes the X-tuple obtained by restricting the mapping to X. A *Y-relation* (or a *relation over Y*, or simply a *relation*, if Y is understood) is a finite set of Y-tuples. If r is a Y-relation and X is a subset of Y, then by $r[X]$, the *projection* of r onto X, we mean the set of all tuples $t[X]$, where t is in r. We shall often denote sets of attributes by uppercase letters and relations by lowercase letters.

If $\mathcal{N}$ is a set of attributes, then we define a *database scheme* $\mathbf{R} = \{R_1, \ldots, R_n\}$ to be a set of subsets of $\mathcal{N}$. Intuitively, for each i, the set R_i of attributes is considered the set of column names for a relation. We may call the R_i's *relation schemes*. If $r_1, \ldots, r_n$ are relations, where r_i is a relation over R_i $(1 \leq i \leq n)$, then we may say that $\mathbf{r} = \{r_1, \ldots, r_n\}$ is a *database over* $\mathbf{R}$. We may call r_i *the R_i relation*.

We have already defined a *hypergraph* to be a pair $(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a set of *nodes* and $\mathcal{E}$ is a set of *edges* (or *hyperedges*) which are arbitrary nonempty subsets of $\mathcal{N}$. We sometimes refer to the edges as "full" edges, to distinguish them from "partial" edges, which we discuss later.

The hypergraph of a database scheme $\{R_1, \ldots, R_n\}$ has as its set of nodes those attributes that appear in one or more of the R_i's, and as its set of edges $\mathbf{R} = \{R_1, \ldots, R_n\}$. We shall often speak of the "hypergraph $\mathbf{R}$" without mentioning the set $\mathcal{N}$ of nodes; then we tacitly assume that $\mathcal{N} = \bigcup \{R_i : 1 \leq i \leq n\}$.

Let us give some terminology for hypergraphs. A *path* from node s to node t is a sequence of $k \geq 1$ edges $E_1, \ldots, E_k$ such that

(i) s is in E_1,
(ii) t is in E_k, and
(iii) $E_i \cap E_{i+1}$ is nonempty if $1 \leq i < k$.

We also say that the above sequence of edges is a path from E_1 to E_k.

Two nodes (or attributes) are *connected* if there is a path from one to the other. Similarly, two edges are connected if there is a path from one to the other. A set of nodes or edges is connected if every pair is connected. A *connected component* is a maximal connected set of edges.

Let $(\mathcal{N}, \mathcal{E})$ be a hypergraph. Its *reduction* $(\mathcal{N}, \mathcal{E}')$ is obtained by removing from $\mathcal{E}$ each edge that is a proper subset of another edge. A hypergraph is reduced if it equals its reduction, that is, if no edge is a subset of another edge.

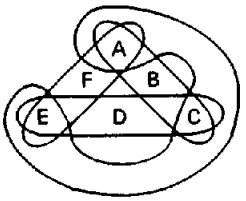


FIGURE 3

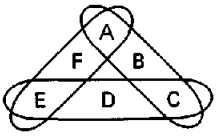


FIGURE 4

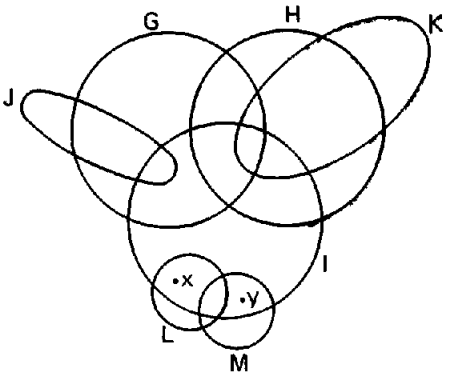


FIGURE 5

Let $\mathcal{M}$ be a set of nodes of the hypergraph $(\mathcal{N}, \mathcal{E})$. The *set of partial edges generated by* $\mathcal{M}$ is defined to be obtained by intersecting the edges in $\mathcal{E}$ with $\mathcal{M}$, that is, taking $\{E \cap \mathcal{M} : E \in \mathcal{E}\} - \{\emptyset\}$ and then taking the reduction of this set of edges. The set of partial edges generated from $(\mathcal{N}, \mathcal{E})$ by some set $\mathcal{M}$ is said to be a *node-generated set of partial edges.*

Let $\mathcal{F}$ be a connected, reduced set of partial edges, and let E and F be in $\mathcal{F}$. Let $Q = E \cap F$. We say that Q is an *articulation set* of $\mathcal{F}$ if the result of removing Q from every edge of $\mathcal{F}$, that is, $\{E - Q : E \in \mathcal{F}\} - \{\emptyset\}$, is not a connected set of partial edges. It is clear that an articulation set in a hypergraph is a generalization of the concept of an articulation point in an ordinary graph.

A *block* of a reduced hypergraph is a connected, node-generated set of partial edges with no articulation set. A set is *trivial* if it contains less than two members. A reduced hypergraph is *α-acyclic* if all its blocks are trivial; otherwise, it is *α-cyclic*. A hypergraph is said to be α-cyclic or α-acyclic precisely if its reduction is.

Example 2.1. It is straightforward to verify that Figure 3 shows an α-acyclic hypergraph. Its edges are ABC, CDE, EFA, and ACE. (We follow the usual database convention that $\{A, B, C\}$ is abbreviated by ABC, etc.) An articulation set for the set of all edges is $ABC \cap ACE = AC$, since the result of removing A and C from each edge is to leave the set of edges B, DE, EF, and E, which is not connected (B is disconnected from the others). Note that the subset $\{ABC, CDE, EFA\}$ of the edges (Figure 4) has no articulation set. However, this set is not node-generated, so there is no contradiction of our assertion that the hypergraph of Figure 3 is α-acyclic. □

Let $(\mathcal{N}, \mathcal{E})$ be a hypergraph, and let $\mathcal{F}$ be a subset of $\mathcal{E}$. Let $\mathcal{M}$ be the set of nodes that is the union of members in $\mathcal{F}$. We say that $\mathcal{F}$ is *closed* if for each edge E of the hypergraph, there is an edge F in $\mathcal{F}$ such that $E \cap \mathcal{M} \subseteq F$. For example, $\{G, H, I\}$ in Figure 5 is a closed set of edges. Thus the intersection of edge K with $G \cup H \cup I$ is contained in edge H; similarly, the intersection of edge J with $G \cup H \cup I$ is contained in G, and the intersection of each of edges L and M with $G \cup H \cup I$ is contained in I. However, $\{L, M\}$ is *not* a closed set of edges, if nodes x and y are present, as drawn in Figure 5. For, the intersection of edge I with $L \cup M$ is contained in neither L nor M. Note that every closed set of (full) edges is always a node-generated set of partial edges. Note also that every set of edges in an ordinary undirected graph is automatically closed.

Recall that a reduced hypergraph is α-acyclic if every nontrivial, connected, node-generated set of partial edges has an articulation set. It follows from results of Fagin

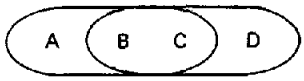


FIGURE 6

et al. [22] that a reduced hypergraph is α-acyclic if and only if every nontrivial, connected, *closed* set of (full) edges has an articulation set. We make use of this characterization later.

A database scheme **R** is said to α-acyclic (respectively, α-cyclic) precisely if the corresponding hypergraph is. Every α-acyclic database scheme has a number of desirable properties, each of which is equivalent to α-acyclicity [7, 8, 22, 23, 25, 32]. We discuss some of these properties in later sections.

3. *Berge-acyclicity*

We now present Berge's [10] concept of acyclicity. A *Berge cycle* in a hypergraph $\mathscr{H}$ is a sequence $(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1})$ such that

(i) $x_1, \ldots, x_m$ are distinct nodes of $\mathscr{H}$;
(ii) $S_1, \ldots, S_m$ are distinct edges of $\mathscr{H}$, and $S_{m+1} = S_1$;
(iii) $m \geq 2$, that is, there are at least 2 edges involved; and
(iv) x_i is in S_i and S_{i+1} $(1 \leq i \leq m)$.

A hypergraph is *Berge-cyclic* if it has a Berge cycle; otherwise, it is *Berge-acyclic.*

As an example, the hypergraph of Figure 6 is Berge-cyclic, because it contains the Berge cycle $(\underline{ABC}, C, \underline{BCD}, B, \underline{ABC})$, where, for clarity, the edges are underlined. We see from this example that if some pair of edges of a hypergraph have two or more nodes in common, then the hypergraph is Berge-cyclic.

4. *β-acyclicity*

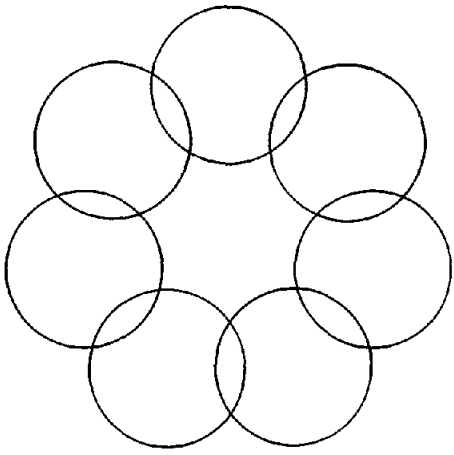
In this section we give various definitions for another degree of acyclicity, called β-acyclicity. We show that the definitions are equivalent. One of these definitions says that a hypergraph **R** is β-acyclic if and only if every subhypergraph of **R** is α-acyclic (if $\mathbf{S} \subseteq \mathbf{R}$ is a subset, not necessarily proper, of the edges **R**, then **S** is a *subhypergraph* of **R**.) Thus, although the hypergraph in Figure 3 is α-acyclic, it is not β-acyclic, because the subhypergraph in Figure 4 is α-cyclic.

Because of this characterization of β-acyclic hypergraphs, and because of the importance of α-acyclic database schemes [4, 7, 8, 14, 22, 23, 25, 29, 32, 38], it follows that β-acyclic database schemes are also important. For it is very natural to deal with subschemes of a relational database scheme. Thus a database scheme is β-acyclic if and only if every subscheme is α-acyclic.

Properties of α-acyclic schemes "relativize" to β-acyclic schemes, as we mentioned in the introduction. Thus, if $\mathscr{P}$ is one of the various desirable properties of database schemes that is equivalent to α-acyclicity, then a database scheme is β-acyclic if and only if every one of its subschemes enjoys property $\mathscr{P}$. It is informative to give an example.

A database scheme is α-acyclic if and only if there is a semijoin program that can assist a user who is interested in taking a join over *all* of the relations in the database. By "assist a user" we mean that the semijoin program converts the original database into a (globally) consistent database (for details and definitions, see [8] or [11]). Therefore, a database scheme is β-acyclic if and only if *no matter what subset* of the relations in the database the user wants to join, there is a semijoin program that can assist him.

FIGURE 7



We now prepare to give our various definitions of β-acyclicity. Actually, it is convenient instead to define *β-cyclicity*. Of course, we say that a hypergraph is *β-acyclic* if and only if it is not β-cyclic. A database scheme is β-acyclic (respectively, β-cyclic) precisely if the corresponding hypergraph is.

Let $(S_1, \ldots, S_m, S_{m+1})$ be a sequence of sets, where $S_1, \ldots, S_m$ are distinct and $S_{m+1} = S_1$. Let us call S_i and S_{i+1} *neighbors* $(1 \leq i \leq m)$; note, in particular, that S_m and S_1 are neighbors. Let us call $(S_1, \ldots, S_m, S_{m+1})$ a *pure cycle* if $m \geq 3$ (i.e., at least three sets are involved) and if whenever $i \neq j$, then $S_i \cap S_j$ is nonempty if and only if S_i and S_j are neighbors. Thus a pair is nondisjoint precisely if it is a neighboring pair. Furthermore, if $m = 3$, then we assume also that $S_1 \cap S_2 \cap S_3$ is empty. If $m \geq 4$, then the comparable assumption (i.e., the assumption that $S_1 \cap \cdots \cap S_m$ is empty) is unnecessary, since it is a consequence of our other assumptions. A pure cycle with seven edges appears in Figure 7, where two edges have nonempty intersection if and only if they are shown to intersect in Figure 7. Of the types of cycles for hypergraphs which we discuss in this paper, a pure cycle is certainly the most natural and noncontroversial. (However, Kahn et al. [30] have defined several notions of acyclicity for hypergraphs for which a pure cycle may be an acyclic hypergraph!)

A *β-cycle* in a hypergraph $\mathscr{H}$ is a sequence $(S_1, \ldots, S_m, S_{m+1})$ of edges such that if $X = S_1 \cap \cdots \cap S_m$, and S_i' is the set difference $S_i - X$ $(1 \leq i \leq m)$, then $(S_1', \ldots, S_m', S_{m+1}')$ is a pure cycle. Thus every β-cycle is of the form $(S_1' \cup X, \ldots, S_m' \cup X, S_{m+1}' \cup X)$, where $(S_1', \ldots, S_m', S_{m+1}')$ is a pure cycle. S_i' need not be an edge of the hypergraph, although S_i is $(1 \leq i \leq m + 1)$. Of course, every pure cycle of edges is also a β-cycle.

We are now ready to give our first three definitions of β-cyclicity (we shall give five definitions altogether).

Definition 1. A hypergraph is β-cyclic if it has a β-cycle.

Definition 2. A hypergraph is β-cyclic if some subhypergraph is α-cyclic.

Definition 3. A hypergraph is β-cyclic if some nontrivial, connected, reduced set of edges has no articulation set.

In Definition 3 we can replace "nontrivial, connected, reduced set of edges" (which means "connected, reduced set of at least two edges") by "connected, reduced set of at least *three* edges" and get an equivalent definition. This is because every connected,

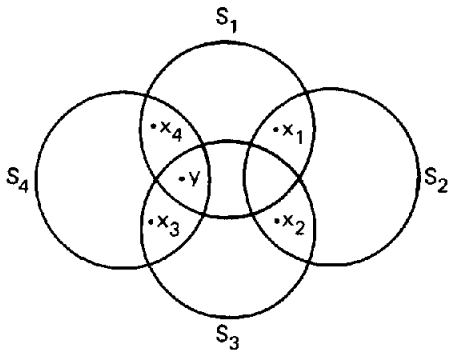


FIGURE 8

reduced set of two edges clearly has an articulation set. Further, if we work only with *reduced* hypergraphs, as is often the case, then we can drop the word "reduced" in Definition 3.

Now a hypergraph is α-cyclic precisely if some nontrivial, connected, reduced, *closed* set of edges has no articulation set (this statement follows immediately from results in [22]). Note that the only difference between this characterization of α-cyclicity and the characterization of β-cyclicity in Definition 3 is that in Definition 3 the word "closed" does not appear. Thus β-cyclicity may be considered a more natural graph-theoretic concept than α-cyclicity, since the somewhat arbitrary concept of closedness is dropped. Further, a hypergraph is α-cyclic precisely if some nontrivial, connected, node-generated set of partial edges has no articulation set. This characterization of α-cyclicity is identical to the characterization of β-cyclicity in Definition 3, except that Definition 3 deals with "reduced sets of edges" rather than with the more complex "node-generated sets of partial edges."

Our next definition of β-cyclicity is given to provide an analogy with Berge-acyclicity and with two of our definitions of γ-cyclicity (Section 5). A *weak β-cycle* in a hypergraph $\mathscr{H}$ is a sequence $(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1})$ such that

(i) $x_1, \ldots, x_m$ are distinct nodes of $\mathscr{H}$;
(ii) $S_1, \ldots, S_m$ are distinct edges of $\mathscr{H}$, and $S_{m+1} = S_1$;
(iii) $m \geq 3$, that is, there are at least 3 edges involved; and
(iv) x_i is in S_i and S_{i+1} $(1 \leq i \leq m)$ and in no other S_j.

It is sometimes convenient to refer to the sequence $(S_1, \ldots, S_m, S_{m+1})$ of edges alone of a weak β-cycle as a weak β-cycle. Under this notation, every β-cycle is clearly a weak β-cycle, but the converse is false, as we shall see. However, it is not hard to see that the *shortest* weak β-cycle in a hypergraph is a β-cycle (we shall prove a stronger result in the proof of Theorem 4.1). Note that if we change "3" everywhere in (iii) to "2" and drop "and in no other S_j" in (iv), then we get the definition of a Berge cycle.

The sequence $(S_1, x_1, S_2, x_2, S_3, x_3, S_4, x_4, S_1)$ in Figure 8 is a weak β-cycle. However, it is not a β-cycle, because the node y is in S_1, S_3, and S_4 but not in S_2.

Definition 4. A hypergraph is β-cyclic if it has a weak β-cycle.

Our final definition of β-cyclicity is essentially due to Graham [26]. Let $(S_1, \ldots, S_m, S_{m+1})$ be a sequence of edges, where $S_1, \ldots, S_m$ are distinct and $S_{m+1} = S_1$. Assume further that $m \geq 3$ (i.e., that at least three edges are involved). Define $\Delta_i = S_i \cap S_{i+1}$ $(1 \leq i \leq m)$. We say that $(S_1, \ldots, S_m, S_{m+1})$ is a *Graham cycle* if each Δ_i is nonempty $(1 \leq i \leq m)$ and whenever $i \neq j$, then Δ_i and Δ_j are incomparable (i.e., $\Delta_i \not\subseteq \Delta_j$ and $\Delta_j \not\subseteq \Delta_i$). Graham calls a hypergraph *CAG-C* if it has no Graham cycle.

Definition 5. A hypergraph is β-cyclic if it has a Graham cycle.

It is clear that every weak β-cycle is a Graham cycle.

THEOREM 4.1. *Definitions* 1–5 *of* β*-cyclicity are equivalent.*

PROOF. We show that (3) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (4) $\Rightarrow$ (5) $\Rightarrow$ (1) $\Rightarrow$ (3). By "(i) $\Rightarrow$ (j)" we mean that every hypergraph that is β-cyclic by definition (i) is β-cyclic by definition (j).

(3) $\Rightarrow$ (2): Let $\mathscr{H}$ be β-cyclic by Definition 3. By Definition 3, $\mathscr{H}$ has a nontrivial, connected, reduced set $\mathbf{E}$ of edges with no articulation set. Then $\mathbf{E}$ is an α-cyclic hypergraph, since the set $\mathbf{E}$ of edges is a node-generated set of partial edges in the hypergraph $\mathbf{E}$. Hence $\mathscr{H}$ is β-cyclic by Definition 2.

(2) $\Rightarrow$ (3): Let $\mathscr{H}$ be a β-cyclic hypergraph by Definition 2; we shall show that it is β-cyclic by Definition 3. By Definition 2, $\mathscr{H}$ has a subhypergraph $\mathscr{F}$ that is α-cyclic. Let $\mathscr{F}'$ be the reduction of $\mathscr{F}$. Then $\mathscr{F}'$ is a reduced, α-cyclic subhypergraph of $\mathscr{H}$ (recall that a hypergraph is α-cyclic precisely if its reduction is). Thus $\mathscr{F}'$ has a nontrivial, connected set $\mathbf{E}$ of edges with no articulation set (the set $\mathbf{E}$ is also closed in $\mathscr{F}'$, although we do not need this fact). Clearly $\mathbf{E}$ is reduced, since $\mathscr{F}'$ is. Hence, $\mathscr{H}$ is β-cyclic, by Definition 3.

(3) $\Rightarrow$ (4): Let $\mathscr{H}$ be β-cyclic by Definition 3; we shall show that it is β-cyclic by Definition 4. Since $\mathscr{H}$ is β-cyclic by Definition 3, it has a connected, reduced set $\mathbf{E}$ of at least three edges and with no articulation set. (See the comment following Definition 3.) Find two distinct edges V and W in $\mathbf{E}$ such that the number of nodes in $V \cap W$ is as big as possible. Since $\mathbf{E}$ is connected, we know that some pair of edges in $\mathbf{E}$ has nonempty intersection, and so V and W also have nonempty intersection. Let us denote $V \cap W$ by Q. We know that Q is a proper subset of each of V and W, because $\mathbf{E}$ is reduced. Since $\mathbf{E}$ has no articulation set, we know that the result of removing Q from every edge in $\mathbf{E}$ leaves a connected set of partial edges. Hence there is a sequence $(S_1, \ldots, S_k)$ of distinct edges in $\mathbf{E}$ for which

(i) $S_1 = V$,
(ii) $S_k = W$,
(iii) $(S_i \cap S_{i+1}) - Q$ is nonempty for $1 \leq i < k$.

Let us choose the sequence $(S_1, \ldots, S_k)$ as above so that k is as small as possible. Since $(S_i \cap S_{i+1}) - Q$ is nonempty by (iii), it contains a node x_i $(1 \leq i < k)$. If j is not i or $i+1$, then x_i is not in S_j; otherwise the sequence $(S_1, \ldots, S_k)$ could be shortened and still maintain properties (i)–(iii) above, and this would violate minimality of k. Hence, if we can find m with $3 \leq m \leq k$ (and so, in particular, $3 \leq k$) such that S_1 and S_m contain a node v that is not in S_j for $1 < j < m$, then $(S_1, x_1, S_2, x_2, \ldots, S_m, v, S_1)$ is a weak β-cycle (where we are using the edge-node-edge notation for clarity), and we are done.

Now $Q \not\subseteq S_1 \cap S_2$. For if $Q \subseteq S_1 \cap S_2$, then $S_1 \cap S_2$ would have strictly more nodes than Q, since $(S_1 \cap S_2) - Q$ is nonempty by (iii) above. However, this would contradict the maximality of Q (recall that V and W were chosen in $\mathbf{E}$ such that $Q = V \cap W$ is as big as possible).

Since $Q \not\subseteq S_1 \cap S_2$, let v be a node in Q that is not in $S_1 \cap S_2$. Now $v \in S_1$ (since $Q \subseteq V = S_1$). Hence, since $v \notin S_1 \cap S_2$, it follows that $v \notin S_2$. Let m be minimal such that $3 \leq m \leq k$ and $v \in S_m$. There is such an m, since $v \in S_k$ (because $v \in Q \subseteq W = S_k$). Then S_1 and S_m contain the node v, which is not in S_j for $1 < j < m$. This was to be shown.

(4) ⇒ (5): Let $\mathscr{H}$ be β-cyclic by Definition 4; it is then β-cyclic by Definition 5, since, as we noted, every weak β-cycle is a Graham cycle.

(5) ⇒ (1): Let $\mathscr{H}$ be β-cyclic by Definition 5; we shall show that it is β-cyclic by Definition 1. Let $\mathscr{S} = (S_1, \ldots, S_m, S_{m+1})$ be a minimal Graham cycle, that is, a Graham cycle with m, the number of edges in the Graham cycle, as small as possible. We shall show that $\mathscr{S}$ is a β-cycle, which shows that $\mathscr{H}$ is β-cyclic by Definition 1. Let $X = S_1 \cap \cdots \cap S_m$, and let S_i' be the set difference $S_i - X$ $(1 \leq i \leq m)$. We must show that $\mathscr{S}' = (S_1', \ldots, S_m', S_{m+1}')$ is a pure cycle.

Let Δ_i (respectively, Δ_i') be $S_i \cap S_{i+1}$ (respectively, $S_i' \cap S_{i+1}'$), for $1 \leq i \leq m$. Each Δ_i' $(1 \leq i \leq m)$ is nonempty. For if Δ_i' were empty, then $\Delta_i \subseteq \Delta_j$ for each j, and in particular, for some $j \neq i$; this contradicts our assumption that $\mathscr{S}$ is a Graham cycle.

We have shown that each pair of neighbors in $\mathscr{S}'$ is nondisjoint (since each Δ_i' is nonempty). By construction we know that $S_1' \cap \cdots \cap S_m'$ is empty. To show that $\mathscr{S}'$ is a pure cycle, we need only show now that nonneighbors are disjoint. Assume not; we shall derive a contradiction. Let S_p' and S_q' be nonneighbors that are nondisjoint. Take a node v in $S_p' \cap S_q'$. By construction of $\mathscr{S}'$, we know that $v \notin S_r'$ for some r. By interchanging the roles of S_p' and S_q', if necessary, we can assume that proceeding "clockwise" on $\mathscr{S}'$ from S_p' to S_q', we encounter S_r' on the way. (The "clockwise" direction is from S_1' to S_2' to ... to S_m' to S_1' to ....) Consider the following conditions on a pair (s, j) of indices:

(a) S_s' and S_j' are distinct and nonneighbors, and
(b) there is a node w in $S_s' \cap S_j'$ that is not in some S_k' that lies on the clockwise path from S_s' to S_j'.

These conditions can be fulfilled by letting w, s, j, and k be, respectively, v, p, q, and r. Select s and j such that (a) and (b) are satisfied, and such that the clockwise path from S_s' to S_j' is as short as possible. By doing a cyclic shift of the subscripts, if necessary, we can assume that $s = 1$. Thus, $1 < j < m$, and the node $w \in S_1' \cap S_j'$, but $w \notin S_k'$ for some k, with $1 < k < j$. Also, $j \geq 3$, since S_1' and S_j' are not neighbors. We now show that for each p with $1 < p < j$, necessarily $w \notin S_p'$. For, assume $w \in S_p'$ and $1 < p < j$. There are two cases, depending on whether $p < k$ or $k < p$. Assume $p < k$; the other case is similar. Then there is a node (namely, w) in S_p' and S_j' but not in S_k', and also S_k' is on the clockwise path from S_p' to S_j' (see Figure 9). Since the clockwise path from S_p' to S_j' is shorter than the clockwise path from S_1' to S_j', this contradicts our minimality assumption in the choice of s and j. Hence $w \notin S_p'$ whenever $1 < p < j$.

We now show that

$$(S_1, \ldots, S_j, S_1) \tag{4.1}$$

is a Graham cycle. Let Δ_i and Δ_i' be as before $(1 \leq i < j)$, let Δ be $S_j \cap S_1$, and let Δ' be $S_j' \cap S_1'$. We already know that Δ_i and Δ_j are pairwise incomparable when $i \neq j$, since $(S_1, \ldots, S_m, S_1)$ is a Graham cycle and $j < m$. Further, Δ and each Δ_i are nonempty. Hence, to show that (4.1) is a Graham cycle, we need only show that Δ is incomparable with each of the Δ_i's. Now Δ contains w, which is not in any of the Δ_i's. Thus, $\Delta \not\subseteq \Delta_i$ $(1 \leq i < j)$. So, to show that (4.1) is a Graham cycle, we need only show that $\Delta_i \not\subseteq \Delta$ $(1 \leq i < j)$. Assume not; we shall derive a contradiction. Find n $(1 \leq n < j)$ such that $\Delta_n \subseteq \Delta$. Therefore, since $\Delta_n' = \Delta_n - X$ and $\Delta' = \Delta - X$, it follows that

$$\Delta_n' \subseteq \Delta'. \tag{4.2}$$

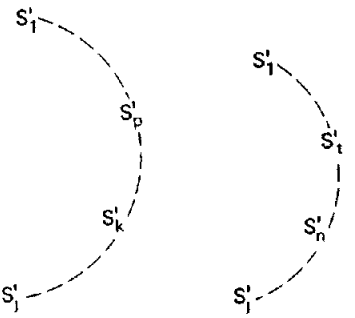


FIGURE 9

FIGURE 10

Let x be an arbitrary member of Δ'_n. By (4.2), we know that $x \in \Delta'$. We now show that

$$x \in S'_i \quad \text{for} \quad 1 \leq i \leq j. \tag{4.3}$$

Assume not. Find t $(1 \leq t \leq j)$ such that $x \notin S'_t$. Since $x \in \Delta'_n$ and $x \in \Delta'$, we know that t is not any of 1, n, $n+1$, or j. There are now two cases, depending on whether $t < n$ or $n + 1 < t$. Assume $t < n$; the other case is similar. Then (see Figure 10)

(a) S'_1 and S'_n are distinct and nonneighbors, and
(b) node x is in $S'_1 \cap S'_n$ but not in S'_t, and S'_t lies on the clockwise path from S'_1 to S'_n.

But the clockwise path from S'_1 to S'_n is strictly shorter than the clockwise path from S'_1 to S'_j. This contradicts our minimality assumption in our choice of s and j. This proves (4.3). Since $S'_i \subseteq S_i$, it follows from (4.3) that $x \in S_i$, for $1 \leq i \leq j$. Therefore, $x \in \Delta_i$ $(1 \leq i < j)$. But x was an arbitrary member of Δ'_n. Hence, $\Delta'_n \subseteq \Delta_i$ $(1 \leq i < j)$. Thus, $\Delta_n \subseteq \Delta_i$ $(1 \leq i < j)$, since the only nodes in Δ_n that are not in Δ'_n are the nodes X that are in every S_i. Let a be arbitrary such that $1 \leq a < j$ and $a \neq n$. There is such an a, since $j \geq 3$. Since $\Delta_n \subseteq \Delta_a$, this contradicts the fact that the Δ_i's are pairwise incomparable. This contradiction establishes our claim that (4.1) is a Graham cycle. But (4.1) is a shorter Graham cycle than our allegedly smallest Graham cycle $(S_1, \ldots, S_m, S_1)$. This contradiction shows that the smallest Graham cycle is indeed a β-cycle, which was to be shown. (As a matter of interest, we note that although the *smallest* Graham cycle is always a β-cycle, there may be a Graham cycle that is not even a weak β-cycle.)

(1) $\Rightarrow$ (3): Let $\mathscr{H}$ be β-cyclic by Definition 1. Therefore, it has a β-cycle $(S_1, \ldots, S_m, S_{m+1})$, where $m \geq 3$. The set $\{S_1, \ldots, S_m\}$ of edges is clearly a nontrivial, connected, reduced set of edges with no articulation set. So $\mathscr{H}$ is β-cyclic by Definition 3. □

We note that recently Graham [27] has independently shown the equivalence of Definitions 2 and 5.

5. γ-*acyclicity*

As in the case of β-cyclicity, we shall give several equivalent definitions of γ-cyclicity. A hypergraph is γ-*acyclic* if it is not γ-cyclic. A database scheme is γ-acyclic (respectively, γ-cyclic) precisely if the corresponding hypergraph is.

A γ-*cycle* in a hypergraph $\mathscr{H}$ is a sequence

$$(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1}) \tag{5.1}$$

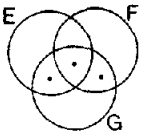


FIGURE 11

such that

(i) $x_1, \ldots, x_m$ are distinct nodes of $\mathscr{H}$;
(ii) $S_1, \ldots, S_m$ are distinct edges of $\mathscr{H}$, and $S_{m+1} = S_1$;
(iii) $m \geq 3$, that is, there are at least 3 edges involved;
(iv) x_i is in S_i and S_{i+1} $(1 \leq i \leq m)$; and
(v) if $1 \leq i < m$, then x_i is in no S_j except S_i and S_{i+1}.

Note that the only difference between a γ-cycle and a weak β-cycle is that "$1 \leq i < m$" in (v) is replaced by "$1 \leq i \leq m$" to define a weak β-cycle. Thus every weak β-cycle is a γ-cycle. Note also that the only difference between a γ-cycle and a Berge-cycle is that to define a Berge cycle, "3" is replaced everywhere in (iii) by "2," and also (v) is dropped. As before, it is sometimes convenient to refer to the sequence $(S_1, \ldots, S_m, S_{m+1})$ of edges alone of a γ-cycle as a γ-cycle. We say that this γ-cycle, with m distinct edges, is of *size m*.

Definition 1. A hypergraph is γ-cyclic if it has a γ-cycle.

We define a *weak γ-cycle* just as we defined a γ-cycle, except that "$1 \leq i < m$" in (v) is replaced by "$i = 1$ or $i = 2$" to define a weak γ-cycle. Thus every γ-cycle is a weak γ-cycle. Although the converse is false, it is true that the *shortest* weak γ-cycle in a hypergraph is a γ-cycle (we shall prove a stronger result in the proof of Theorem 5.1 below).

Definition 2. A hypergraph is γ-cyclic if it has a weak γ-cycle.

To help prevent confusion, we note that in an earlier version of this paper we referred to what we are now calling a weak γ-cycle as a γ-cycle.

The next definition gives us a nice characterization of γ-cyclic hypergraphs.

Definition 3. A hypergraph is γ-cyclic if it has either a γ-cycle of size 3 or a pure cycle.

It is easy to see that a hypergraph is γ-cyclic according to Definition 3 precisely if it contains at least one of two kinds of "forbidden configurations" of edges: either a pure cycle, as in Figure 7, or a set of three edges that intersect at least as shown as in Figure 11. (By the latter, we mean that in Figure 11 there is at least one node in $E \cap F \cap G$, there is at least one node in $(E \cap G) - F$, and there is at least one node in $(F \cap G) - E$. Other intersections involving combinations of E, F, and G may also occur.) For, if there is a γ-cycle of size 3, then either there is a configuration as in Figure 11 or else there is a pure cycle of size 3.

Our next definition (Definition 4) of γ-cyclicity is due to Goodman and Shmueli [24], who, after reading an early draft of this paper, pointed out to the author that Definition 4 is equivalent to the author's Definitions 1–3.

Definition 4. A hypergraph is γ-cyclic if it has a pair E, F of incomparable, nondisjoint edges such that in the hypergraph that results by removing $E \cap F$ from every edge, what is left of E is connected to what is left of F.

Remark. We say that E and F are *incomparable* if $E \not\subseteq F$ and $F \not\subseteq E$. Definition 4 says that hypergraph $\mathscr{H}$ is γ-cyclic if it has a pair E, F of incomparable, nondisjoint edges such that if $Q = E \cap F$, if G' is $G - Q$ for each edge G of $\mathscr{H}$, and if $\mathscr{H}' = \{G' : G \text{ is an edge of } \mathscr{H}\} - \{\emptyset\}$, then E' and F' are connected in $\mathscr{H}'$.

There is a pretty algorithm (defined in [18]) for determining γ-acyclicity. It is very similar in flavor to "Graham's algorithm" for determining α-acyclicity. Both of these algorithms will be presented in Section 9.

THEOREM 5.1. *Definitions* 1–4 *of* γ*-cyclicity are equivalent.*

PROOF. We show that (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (4) $\Rightarrow$ (1). By "(i) $\Rightarrow$ (j)", we mean that every hypergraph that is γ-cyclic by definition (i) is γ-cyclic by definition (j).

(1) $\Rightarrow$ (2): This is immediate, since, as noted, every γ-cycle is a weak γ-cycle.

(2) $\Rightarrow$ (3): Let $\mathscr{H}$ be γ-cyclic by Definition 2; we shall show that $\mathscr{H}$ is γ-cyclic by Definition 3. Let (5.1) above be a minimal weak γ-cycle in $\mathscr{H}$ (by *minimal* we mean that m is as small as possible). If $m = 3$, then we are done (since a weak γ-cycle of size 3 is clearly a γ-cycle of size 3). So, assume that $m \geq 4$. We shall show that (5.1) is a pure cycle. We already know that neighbors intersect, so we need only show that nonneighbors do not intersect.

We now show that S_1 does not intersect a nonneighbor. Assume it does. Find k ($3 \leq k < m$) as small as possible so that $S_1 \cap S_k \neq \emptyset$. Take v in $S_1 \cap S_k$. Then $(S_1, x_1, \ldots, S_{k-1}, x_{k-1}, S_k, v, S_1)$ is a smaller weak γ-cycle than (5.1). This is a contradiction.

We now show that S_2 does not intersect a nonneighbor. For, assume that $v \in S_2 \cap S_k$, with $4 \leq k \leq m$. There are now two cases.

Case 1. $v \in S_3$. We know that $v \notin S_1$, since S_1 does not intersect its nonneighbor S_3. Find r as big as possible so that $v \in S_r$. It is then easy to see that $(S_1, x_1, S_2, v, S_r, x_r, \ldots, S_m, x_m, S_1)$ is a smaller weak γ-cycle than (5.1). This is a contradiction.

Case 2. $v \notin S_3$. Find r as small as possible so that $v \in S_r$. It is then easy to see that $(S_r, v, S_2, x_2, S_3, x_3, \ldots, S_r)$ is a smaller weak γ-cycle than (5.1). This is a contradiction.

We have shown that neither S_1 nor S_2 intersects a nonneighbor. Find j as small as possible so that S_j intersects a nonneighbor S_k; say $v \in S_j \cap S_k$. Then $3 \leq j$, and $j + 2 \leq k \leq m$. It is easy to see that $(S_1, x_1, S_2, x_2, \ldots, S_j, v, S_k, \ldots, S_{m+1})$ is a smaller weak γ-cycle than (5.1). This contradiction completes the proof of (2) $\Rightarrow$ (3).

(3) $\Rightarrow$ (4): Let $\mathscr{H}$ be γ-cyclic by Definition 3; we shall show that $\mathscr{H}$ is γ-cyclic by Definition 4. Since $\mathscr{H}$ is γ-cyclic by Definition 3, it has either a γ-cycle of size 3 or a pure cycle. Assume first that $\mathscr{H}$ has a γ-cycle of size 3, and let this γ-cycle be $(S_1, x_1, S_2, x_2, S_3, x_3, S_1)$. It is easy to verify that $\mathscr{H}$ is γ-cyclic by Definition 4, where we let E and F be, respectively, S_1 and S_3. Now assume that $\mathscr{H}$ has a pure cycle. By letting E and F be neighboring edges in the pure cycle, we see once again that $\mathscr{H}$ is γ-cyclic by Definition 4.

(4) $\Rightarrow$ (1): Let $\mathscr{H}$ be γ-cyclic by Definition 4; we shall show that H is γ-cyclic by Definition 1. Take E and F as in Definition 4, and let $Q = E \cap F$. We know that there is a sequence $(S_1, \ldots, S_m)$ of edges such that

(i) $S_1 = E$,
(ii) $S_m = F$, and
(iii) $(S_i \cap S_{i+1}) - Q \neq \emptyset$, for $1 \leq i \leq m - 1$.

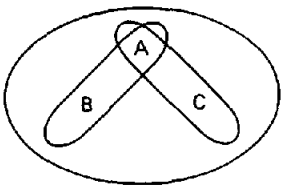


FIGURE 12

Let us also assume that we have selected the S_i's so that (i)–(iii) above hold and m is as small as possible. If $m = 2$, then $S_2 = F$ by (ii), and so $S_1 \cap S_2 = Q$, which contradicts (iii) when $i = 1$. Hence $m \geq 3$. By (iii), we can find a node x_i in $(S_i \cap S_{i+1}) - Q$, for $1 \leq i \leq m - 1$. Define also S_{m+1} to be E ($=S_1$), and define x_m to be a node in $E \cap F$ (by assumption, $E \cap F$ is nonempty). We now show that $(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1})$ is a γ-cycle. The node x_1 is not in any of $S_3, \ldots, S_{m-1}$, by minimality of m (thus, if $x_1 \in S_i$, where $3 \leq i \leq m - 1$, then the sequence $S_1, S_i, S_{i+1}, \ldots, S_m$ could be used in place of $S_1, S_2, \ldots, S_m$). Further, $x_1 \notin S_m = F$, since $x_1 \in E = S_1$ but $x_1 \notin Q = E \cap F$. So x_1 is in S_1 and S_2 but in no other S_j. Similarly, x_i is in S_i and S_{i+1} but in no other S_j, for $1 \leq i \leq m - 1$. In particular, $x_1, \ldots, x_{m-1}$ are all distinct. Further, x_m is distinct from any of $x_1, \ldots, x_{m-1}$, since $x_m \in Q$ but $x_i \notin Q$, for $1 \leq i \leq m$. Thus the nodes $x_1, \ldots, x_m$ are all distinct. The edges $S_1, \ldots, S_m$ are all distinct by minimality of m. We have shown enough to prove that $(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1})$ is a γ-cycle. Hence $\mathcal{H}$ is γ-cyclic by Definition 1, which was to be shown. □

Later we shall identify some desirable properties of database schemes, involving monotone-increasing joins and unique relationships among attributes, such that each of these properties is equivalent to γ-acyclicity.

6. *Relationships Among the Various Degrees of Acyclicity*

We begin by proving the following simple theorem.

THEOREM 6.1. *Berge-acyclicity* $\Rightarrow$ *γ-acyclicity* $\Rightarrow$ *β-acyclicity* $\Rightarrow$ *α-acyclicity. None of the reverse implications hold.*

PROOF. Every α-cyclic hypergraph is β-cyclic, since a hypergraph is β-cyclic if and only if some subhypergraph (including the whole hypergraph itself) is α-cyclic. Also, it is clear from our definitions that every weak β-cycle is a γ-cycle and every γ-cycle is a Berge cycle. It follows that Berge-acyclicity $\Rightarrow$ γ-acyclicity $\Rightarrow$ β-acyclicity $\Rightarrow$ α-acyclicity.

We now show that none of the reverse implications hold. The hypergraph of Figure 3, with edges *ABC*, *CDE*, *EFA*, and *ACE*, is α-acyclic but β-cyclic (since the subhypergraph of Figure 4, with edges *ABC*, *CDE*, and *EFA*, is α-cyclic). The hypergraph of Figure 12, with edges *AB*, *AC*, and *ABC*, is β-acyclic. However, it is γ-cyclic, since ($\underline{AC}$, *C*, $\underline{ABC}$, *B*, $\underline{AB}$, *A*, $\underline{AC}$) is a γ-cycle, where, for clarity, the edges are underlined. The hypergraph of Figure 2 is a *reduced* hypergraph that is β-acyclic but γ-cyclic.

Finally, the hypergraph of Figure 6, with edges *ABC* and *BCD*, is γ-acyclic. However, as we noted in Section 3, it is Berge-cyclic. □

We note that Zaniolo [41] defined two other notions of acyclicity for hypergraphs, in a pioneering effort to find some hypergraph condition that is equivalent to a certain desirable database condition ("every pairwise consistent database is consistent"; see Section 7). Unfortunately, one of his conditions was sufficient but not necessary, and the other was necessary but not sufficient. Neither of his conditions

is equivalent to any of our degrees of acyclicity. We also note that Batini et al. [6] discuss the issue of generating various subclasses of α-acyclic hypergraphs by "hypergraph grammars." Further, Kahn et al. [30] have defined several notions of acyclicity for hypergraphs by generalizing various properties of acyclic graphs.

We now discuss the naturalness of the various degrees of acyclicity, and then we make a few observations contrasting their features.

Berge-acyclicity is too restrictive an assumption to make about database schemes. For, if some pair of distinct relation schemes R_i, R_j in the database scheme $\mathbf{R} = \{R_1, \ldots, R_n\}$ have more than one attribute in common, then $\mathbf{R}$ is Berge-cyclic. For example, the hypergraph of Figure 6, with edges ABC and BCD, is Berge-cyclic, as we noted earlier. A restriction that no two relation schemes can have more than one attribute in common is far too severe. We now show that there are "natural" database schemes that are α-acyclic but β-cyclic, and natural schemes that are β-acyclic but γ-cyclic.

Assume that there are six attributes SUPPLIER, PART, PROJECT, COUNT, DATE, and COST, where SUPPLIERs supply PARTs to PROJECTs; where for each PART and PROJECT, the COUNT tells how many of that PART have been supplied to that PROJECT; where the DATE tells when a given supplier first supplied a given PROJECT; and where the COST is what a given SUPPLIER charges for a given PART. The only constraints are the functional dependencies [15]

$$\{\text{PART, PROJECT}\} \rightarrow \text{COUNT},$$
$$\{\text{SUPPLIER, PROJECT}\} \rightarrow \text{DATE},$$
$$\{\text{SUPPLIER, PART}\} \rightarrow \text{COST},$$

(and their logical consequences). The functional dependency {SUPPLIER, PART} $\rightarrow$ COST says that there is only one COST that a given SUPPLIER charges for a given PART; the SUPPLIER does not, for example, charge different PROJECTs different COSTs for the same PART. By doing a standard decomposition to obtain Boyce–Codd normal form [16], the resulting database scheme has four relation schemes, with attributes, respectively,

{SUPPLIER, PART, PROJECT},
{SUPPLIER, PART, PROJECT},
{SUPPLIER, PART, COST},
{PART, PROJECT, COUNT},
{SUPPLIER, PROJECT, DATE}.

The hypergraph of this scheme is as in Figure 13. But this is just an example of the hypergraph of Figure 2, which is α-acyclic but β-cyclic. To obtain a scheme that is β-acyclic but γ-cyclic, we simply drop the COUNT attribute (and the {PART, PROJECT, COUNT} relation scheme) to obtain the hypergraph of Figure 3. This hypergraph is γ-cyclic, since ({SUPPLIER, PART, COST}, PART, {SUPPLIER, PART, PROJECT}, PROJECT, {SUPPLIER, PROJECT, DATE}, SUPPLIER, {SUPPLIER, PART, COST}) is a γ-cycle.

Although there are natural database schemes that are *γ-cyclic* (such as the example just shown), there are also a number of database schemes that are *γ-acyclic*. (An example appears later in this section, with a demonstration of γ-acyclicity.) Although we should not *demand* γ-acyclicity, it is good to know when a given scheme is γ-acyclic, so that we know that it enjoys the desirable properties discussed in Section 8. Similar comments apply, of course, to β-acyclicity.

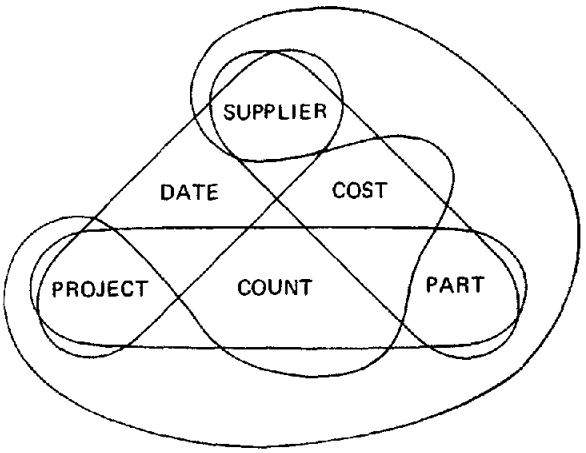


FIGURE 13

As observed in other papers, it *is* natural to demand α-acyclicity; indeed, Fagin et al. [22] and Maier and Ullman [32] argue that α-cyclic schemes represent a possible error in database design. In Section 8 we shall discuss an example of an α-cyclic scheme (which is given in Figure 20) and its "conversion" (by renaming attributes) into a scheme (given in Figure 21) that is not only α-acyclic but even γ-acyclic.

We now contrast some of the features of the various degrees of acyclicity. The proofs of the remarks we now make are straightforward and are left to the reader.

A hypergraph is α-acyclic if and only if its reduction is α-acyclic. However, the analogous statement is false for the other kinds of acyclicity. Thus, the hypergraph of Figure 14 (with edges AB, BC, AC, and ABC) is β-cyclic, γ-cyclic, and Berge-cyclic, although its reduction (which consists of the single edge ABC) is of course acyclic in each of the four senses.

By an *isolated node* we mean as before a node that is in exactly one edge. If $\mathscr{H}$ is a hypergraph and $\mathscr{H}'$ is the result of deleting an isolated node, then $\mathscr{H}$ is θ-acyclic if and only if $\mathscr{H}'$ is θ-acyclic, for $\theta = \alpha$, β, or γ. Although it seems as though the same statement should be true for θ = Berge, there is a subtlety that prevents this. Let $\mathscr{H}$ be the Berge-cyclic hypergraph of Figure 6, with edges ABC and BCD. The result of deleting the isolated nodes A and D is to leave us with two edges, both BC. Since a hypergraph is a *set* of edges (in which there are no duplicates), the resulting hypergraph has only one edge BC and is therefore Berge-acyclic. However, the original hypergraph $\mathscr{H}$ was Berge-cyclic.

By a *singleton edge* we mean an edge with exactly one node, which may or may not be isolated. By a *global node* we mean a node that is in every edge. If $\mathscr{H}'$ is the result of deleting a singleton edge, then $\mathscr{H}$ is θ-acyclic if and only if $\mathscr{H}'$ is θ-acyclic, for $\theta = \alpha$, β, γ, or Berge. If $\mathscr{H}'$ is the result of deleting a global node, then $\mathscr{H}$ is θ-acyclic if and only if $\mathscr{H}'$ is θ-acyclic, for $\theta = \alpha$ or β. The statement is false if $\theta = \gamma$ or Berge. Thus the hypergraph of Figure 12 with edges AB, AC, and ABC, is Berge-cyclic and γ-cyclic. However, the hypergraph of Figure 15, which has edges B, C, and BC and is the result of deleting the global node A, is acyclic in each of the four senses.

We say that two nodes are *edge-equivalent* if they are in precisely the same edges. We shall deal extensively with edge-equivalence in Section 9, where we discuss a polynomial-time algorithm for determining γ-acyclicity. If $\mathscr{H}'$ is the result of deleting a node that is edge-equivalent to another node, then $\mathscr{H}$ is θ-acyclic if and only if $\mathscr{H}'$ is θ-acyclic, for $\theta = \alpha$, β, or γ. The statement is false if θ = Berge. Thus the

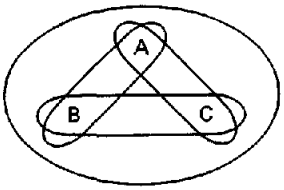


FIGURE 14

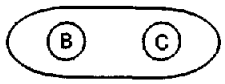


FIGURE 15

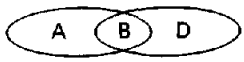


FIGURE 16

hypergraph of Figure 6, with edges ABC and BCD, is Berge-cyclic; however, the hypergraph of Figure 16, which is the result of deleting node C (which is edge-equivalent to B), is Berge-acyclic.

We have just discussed various transformations of hypergraphs and considered whether or not the transformation preserves both θ-acyclicity and θ-cyclicity. That is, we were concerned with the question of whether a hypergraph $\mathscr{H}$ is θ-acyclic if and only if its transform $\mathscr{H}'$ is θ-acyclic. Less restrictively, we might also consider whether or not certain transformations preserve θ-acyclicity (and not be concerned with whether the transformation preserves θ-cyclicity). As an important example, we say that a hypergraph $\mathscr{H}'$ is the result of *uniformly deleting nodes from* $\mathscr{H}$ if there is a set X of nodes of $\mathscr{H}$ such that the edges of $\mathscr{H}'$ are precisely $\{E - X: E \text{ is an edge of } \mathscr{H}\}$. We note that Goodman and Shmueli [23] characterize α-acyclicity in terms of this concept. If $\mathscr{H}'$ is the result of uniformly deleting nodes from $\mathscr{H}$, and $\mathscr{H}$ is θ-acyclic, then so is $\mathscr{H}'$, for $\theta = \alpha$, β, or γ. By the same example as we used in discussing the result of deleting isolated nodes, the statement is false for $\theta =$ Berge. It is easy to see that the result of uniformly deleting nodes from a θ-cyclic hypergraph may be θ-acyclic, for any θ. (For example, every node can be deleted, which leaves the empty hypergraph, a θ-acyclic hypergraph for every θ.)

Another distinction among the various degrees of acyclicity is, as we observed earlier, that a subhypergraph of an α-acyclic hypergraph may be α-cyclic. However, each subhypergraph of a θ-acyclic hypergraph is θ-acyclic, for $\theta = \beta$, γ, or Berge.

As we noted earlier, an ordinary undirected graph is acyclic in the usual sense if and only if it is θ-acyclic when viewed as a hypergraph, for $\theta = \alpha$, β, γ, or Berge. Thus, each of these four concepts of θ-acyclicity is a generalization, from graphs to hypergraphs, of the usual concept of acyclicity.

We close this section by considering the industrial database scheme of [19] and showing that it is γ-acyclic. There are six relation schemes, with attributes, respectively,

{SUPPLIER, PART, PROJECT},
{SUPPLIER, PART, COST},
{EMPLOYEE, SALARY, HIREDATE},
{EMPLOYEE, PROJECT},
{PROJECT, MANAGER},
{SUPPLIER, LOCATION}.

The semantics of this database scheme is explained in [19]. The hypergraph is that of Figure 17. It is easy to verify that by iteratively applying the rules that an isolated node or a singleton edge can be removed (without affecting γ-acyclicity), we are left the empty set. Thus, by the rules of this section, we see that the hypergraph in Figure 17 is γ-acyclic. (In Section 9 we shall give a general polynomial-time algorithm for deciding γ-acyclicity.) However, note that this hypergraph is Berge-cyclic, because

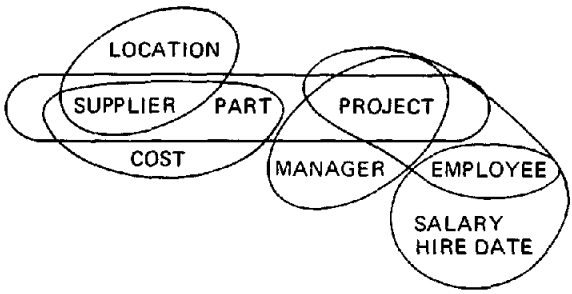


FIGURE 17

FIGURE 18

A	B
0	0
1	1

B	C
0	0
1	1

A	C
0	1
1	0

two edges {SUPPLIER, PART, PROJECT} and {SUPPLIER, PART, COST} share two nodes.

7. *Join Expressions*

Consider the following scenario. A user desires to take the join of four relations r_1, r_2, r_3, and r_4. The following might happen. He might first form $r_1 \bowtie r_2$, which might have, say, a thousand tuples. Then he might join the result with r_3, to obtain $r_1 \bowtie r_2 \bowtie r_3$, a relation with, say, a million tuples. He might finally join the result with r_4, to obtain his desired answer $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$, which might have only ten tuples. Thus, even though the result he was seeking had only ten tuples, he might have had an intermediate result with a million tuples. In this section we discuss "monotone join expressions," which prevent this unpleasant behavior. We first define the important concept of *consistency*.

Let r and s be relations, with attributes R and S, respectively. We say that r and s are *consistent* [8] if $r[R \cap S] = s[R \cap S]$, that is, if the projections of r and s onto their common attributes are the same.

Let $\mathbf{r} = \{r_1, \ldots, r_n\}$ be an arbitrary database over $\mathbf{R} = \{R_1, \ldots, R_n\}$. We say that $\mathbf{r}$ is *pairwise consistent* if r_i and r_j are consistent for each i and j. We say that $\mathbf{r}$ is *globally consistent* (or simply *consistent*) if there is a relation r over attributes $\mathcal{N} = R_1 \cup \cdots \cup R_n$ such that $r_i[R_i] = r[R_i]$ for each i. Thus $\mathbf{r}$ is consistent if there is a "universal relation" r such that each r_i is a projection of r.

It is clear that if $\mathbf{r}$ is consistent, then it is pairwise consistent. If $n = 2$, that is, if only two relations are involved, then it is easy to see that the converse is true. However, in general, the converse is false. For example, let r_1, r_2, and r_3 be the three relations in Figure 18, over attributes AB, BC, and AC, respectively. It is easy to verify that these relations are pairwise consistent but not consistent. Beeri et al. [8] prove that if the database scheme is α-acyclic, then every pairwise consistent database is consistent.

A *join expression* is a well-formed expression formed out of relation schemes, the symbol $\bowtie$, and parentheses, in which every join is binary. For example, if R_1, R_2, R_3, and R_4 are among the relation schemes, then $((R_2 \bowtie R_3) \bowtie (R_1 \bowtie R_4))$ is a join expression, which corresponds to joining the R_2 and the R_3 relations, joining the R_1 and R_4 relations, and then joining together the two results.

Certain join expressions, called sequential join expressions, are of special interest. Let θ be a join expression over $\mathbf{R}$. If θ is of the form $(\cdots ((R_1 \bowtie R_2) \bowtie R_3) \cdots \bowtie R_n)$,

where $R_1, \ldots, R_n$ is an ordering of the distinct members of $\mathbf{R}$, then we say that θ is *sequential*. Intuitively, a sequential join expression $(\cdots ((R_1 \bowtie R_2) \bowtie R_3) \cdots \bowtie R_n)$ corresponds to first joining the R_1 and the R_2 relations, then joining the result with the R_3 relation, then joining the result with the R_4 relation, and so on.

Let θ be a join expression whose relation schemes are all in $\mathbf{R}$, and let $\mathbf{r}$ be a database over $\mathbf{R}$. By $\theta(\mathbf{r})$, we mean the relation that results by replacing each relation scheme R in θ by r, where $r \in \mathbf{r}$ and r has attributes R. For example, if $\mathbf{r} = \{r_1, r_2, r_3, r_4\}$ and θ is the join expression $(R_2 \bowtie (R_3 \bowtie R_2))$, where r_2 and r_3 have attributes R_2 and R_3, respectively, then $\theta(\mathbf{r})$ is the relation $(r_2 \bowtie (r_3 \bowtie r_2))$, that is, the relation $r_2 \bowtie r_3$.

A *subexpression* of a join expression is defined in the usual way. Let θ be a join expression containing relation schemes $\mathbf{R}$, and let $\mathbf{r}$ be a database over $\mathbf{R}$. We say that θ is *monotone with respect to* $\mathbf{r}$ if for every subexpression $(\theta_1 \bowtie \theta_2)$ of θ, the relations $\theta_1(\mathbf{r})$ and $\theta_2(\mathbf{r})$ are consistent. Intuitively, θ is monotone with respect to $\mathbf{r}$ if no tuples are lost in taking any of the binary joins obtained by "executing" $\theta(\mathbf{r})$ as dictated by the parentheses. (We say that *no tuples are lost* in taking the join of relations r and s if r and s are each projections of $r \bowtie s$, i.e., if r and s are consistent.) As an example, $((R_2 \bowtie R_3) \bowtie (R_1 \bowtie R_4))$ is monotone with respect to $\mathbf{r} = \{r_1, r_2, r_3, r_4\}$, where r_i has attributes R_i $(1 \leq i \leq 4)$, if

(a) r_2 and r_3 are consistent,
(b) r_1 and r_4 are consistent, and
(c) $(r_2 \bowtie r_3)$ and $(r_1 \bowtie r_4)$ are consistent.

We say that θ is *monotone* if it is monotone with respect to every pairwise consistent database over $\mathbf{R}$. If θ involves precisely the relation schemes $\mathbf{R}$, then we say that $\mathbf{R}$ *has a monotone join expression.* Monotone join expressions provide an efficient (both space-efficient and time-efficient) manner for taking a join, in that no "intermediate" join has more tuples than the "final" join $r_1 \bowtie \cdots \bowtie r_n$.

Beeri et al. [8] prove the following theorem.

THEOREM 7.1 [8]. *The following are equivalent.*

(1) $\mathbf{R}$ *is* α*-acyclic.*
(2) *There is a monotone join expression over* $\mathbf{R}$.
(3) *There is a monotone, sequential join expression over* $\mathbf{R}$.

Theorem 7.1(3) says that there is an ordering $R_1, \ldots, R_n$ of $\mathbf{R}$ such that if $\mathbf{r} = \{r_1, \ldots, r_n\}$ is a pairwise consistent database over $\mathbf{R}$, then the join $r_1 \bowtie \cdots \bowtie r_i$ is consistent with r_{i+1} $(1 \leq i < n)$. Thus, if we first join r_1 with r_2, join the result with r_3, join the result with r_4, and so on, then no tuples are lost in taking any of the joins; hence the number of tuples grows monotonically. Also, by taking the join in this manner, only one intermediate join needs to be maintained.

We say that a join expression θ is *connected* if for each of its subexpressions $(\theta_1 \bowtie \theta_2)$, there is an attribute that appears in both θ_1 and θ_2. Intuitively, a join expression is connected if none of the binary joins of which it is composed is actually a Cartesian product.

Let us now restrict our attention to database schemes $\mathbf{R}$ for which the corresponding hypergraph is connected. We close this section by showing that every monotone join expression over $\mathbf{R}$ is connected. In the next section we show (among other things) that $\mathbf{R}$ is γ-acyclic if and only if the converse holds, that is, if and only if every connected join expression over $\mathbf{R}$ is monotone.

THEOREM 7.2. *Let* **R** *be a connected hypergraph. Then every monotone join expression over* **R** *is connected.*

PROOF. Let θ be a join expression over **R** that is not connected; we shall show that θ is not monotone. Let r be a relation with attributes $R_1 \cup \cdots \cup R_n$ and with exactly two tuples: a tuple of all 0's and a tuple of all 1's. Let $\mathbf{r} = \{r_1, \ldots, r_n\}$ be a database over $\mathbf{R} = \{R_1, \ldots, R_n\}$, where $r_i = r[R_i]$ for each i. So, **r** is consistent (and hence pairwise consistent).

Since θ is not a connected join expression, it has a subexpression $\delta = \theta_1 \bowtie \theta_2$ such that the attributes of θ_1 are disjoint from the attributes of θ_2. Now $\theta_1(\mathbf{r})$ and $\theta_2(\mathbf{r})$ each have at least two tuples, namely, a tuple of all 0's and a tuple of all 1's. Since $\delta(\mathbf{r}) = \theta_1(\mathbf{r}) \bowtie \theta_2(\mathbf{r})$ is the Cartesian product of $\theta_1(\mathbf{r})$ and $\theta_2(\mathbf{r})$, it follows that $\delta(\mathbf{r})$ has at least four tuples. We shall soon show that $\theta(\mathbf{r}) = r$. Hence, $\theta(\mathbf{r})$ has exactly two tuples, while $\delta(\mathbf{r})$ has at least four tuples. Since δ is a subexpression of θ, it follows that θ is not a monotone join expression, which was to be shown.

Thus, we need only show that $\theta(\mathbf{r}) = r$. Now $\theta(\mathbf{r}) = r_1 \bowtie \cdots \bowtie r_n$, since θ is a join expression over **R**. So, the proof is complete once we show that $r_1 \bowtie \cdots \bowtie r_n = r$. Clearly $r \subseteq r_1 \bowtie \cdots \bowtie r_n$, since each r_i is a projection of r. We now prove the opposite inclusion, that is, that

$$r_1 \bowtie \cdots \bowtie r_n \subseteq r. \tag{7.1}$$

Let u be a tuple in $r_1 \bowtie \cdots \bowtie r_n$; we must show that u is a tuple in r. Since u is in $r_1 \bowtie \cdots \bowtie r_n$, we know that $u[R_i]$ is in r_i, for $1 \le i \le n$. But $r_i = r[R_i]$, so $u[R_i]$ is in $r[R_i]$, for $1 \le i \le n$. This means that there is a tuple q_i of r such that $u[R_i] = q_i[R_i]$, for $1 \le i \le n$. We shall show that all of the q_i's are equal. It then follows that u equals their common value. This implies that u is in r, which proves (7.1).

So, to prove (7.1), we need only show that $q_i = q_j$ for each i and j. Since $\mathbf{R} = \{R_1, \ldots, R_n\}$ is connected, there is a path from R_i to R_j. Therefore, to show that $q_i = q_j$, it is sufficient to show that whenever R_s and R_t are nondisjoint, then $q_s = q_t$. For then, by induction on the length of the path from R_i to R_j, we see that $q_i = q_j$.

Assume now that R_s and R_t are nondisjoint; say $A \in R_s \cap R_t$. Then $q_s[A] = u[A] = q_t[A]$. So q_s and q_t are two tuples of r that agree on an attribute, namely, A. It follows from the definition of r that q_s and q_t are therefore equal. This was to be shown. □

8. *Properties of γ-acyclic Database Schemes*

In this section we discuss several desirable properties for a relational database scheme **R**. Each of these properties is equivalent to the scheme **R** being γ-acyclic. For simplicity, we restrict our attention in this section to database schemes **R** with a connected hypergraph. This restriction is not essential.

(1) **R** *is γ-acyclic.* By this, of course, we mean that the hypergraph of the database scheme is γ-acyclic.

(2) *Every connected join expression over* **R** *is monotone.* Assume that **R** is connected. The equivalence of this property (call it property (2)) with γ-acyclicity is of interest because of the close analogy with Theorem 7.1. Thus, Theorem 7.1 says that **R** is α-acyclic if and only if *some* join expression over **R** is monotone; the equivalence of γ-acyclicity with property (2) (almost) says that **R** is γ-acyclic if and only if *every* join expression over **R** is monotone. We must say "almost" in the previous sentence

FIGURE 19

A	B	C	D
0	1	0	0
0	2	3	4
5	1	3	0

because we actually restrict our attention to connected join expressions. By Theorem 7.2, this is not really a restriction, since the only join expressions that can be monotone are connected.

Property (2) guarantees a great deal of freedom in taking joins. Thus, let $\mathbf{r}$ be a pairwise consistent database over a scheme that obeys property (2). Assume that the user wishes to take a join of some subset of the relations in the database. Property (2) guarantees that he can take his join however he wishes (i.e., he can use whatever join expression he wishes that involves the right relations), and as long as he does not act "foolishly," then he is guaranteed that he is acting in an efficient manner. By "never acting foolishly" we mean that he never joins two relations together whose attributes are disjoint, that is, he never takes a Cartesian product. By "efficient" we mean, as before, that no intermediate join has more tuples than the final join. His choice of how to take the join, that is, which join expression to use, can be dictated by other performance considerations, such as the presence of indices that might speed up the process.

(3) *Every connected, sequential join expression over* $\mathbf{R}$ *is monotone.* Property (3) is to property (2) as Theorem 7.1(3) is to Theorem 7.1(2).

(4) *The join dependency* $\bowtie\mathbf{R}$ *implies that every connected subset of* $\mathbf{R}$ *has a lossless join.* We say [1, 34] that a relation r with attributes $R_1 \cup \cdots \cup R_n$ obeys the *join dependency* $\bowtie\{R_1, \ldots, R_n\}$ if $r = \bowtie\{r_1, \ldots, r_n\}$, where $r_i = r[R_i]$, for $1 \leq i \leq n$. It follows that the join dependency $\bowtie\{R_1, \ldots, R_n\}$ holds for the relation r if and only if r contains each tuple t for which there are tuples $w_1, \ldots, w_n$ of r (not necessarily distinct) such that $w_i[R_i] = t[R_i]$ for each i $(1 \leq i \leq n)$. As an example, the relation r in Figure 19 violates the join dependency $\bowtie\{AB, ACD, BC\}$. For, let w_1, w_2, and w_3 be, respectively, the tuples (0, 1, 0, 0), (0, 2, 3, 4), and (5, 1, 3, 0) of r; let R_1, R_2, and R_3 be, respectively, AB, ACD, and BC; and let t be the tuple (0, 1, 3, 4); then $w_i[R_i] = t[R_i]$ for each i $(1 \leq i \leq n)$, although t is not a tuple in the relation r. However, it is straightforward to verify that the same relation r obeys, for example, the join dependency $\bowtie\{ABC, BCD, ABD\}$.

Let $\mathbf{S} = \{S_1, \ldots, S_m\}$. If $S_1 \cup \cdots \cup S_m$ is a subset of the attributes of the relation r, then we say that r obeys the *embedded join dependency* $\bowtie\mathbf{S}$ if the projection $r[S_1 \cup \cdots \cup S_m]$ obeys the join dependency $\bowtie\mathbf{S}$. When we say that a set $\{S_1, \ldots, S_m\}$ *has a lossless join*, we mean that the embedded join dependency $\bowtie\mathbf{S}$ holds. Thus, property (4) says that every relation that obeys the join dependency $\bowtie\mathbf{R}$ also obeys the embedded join dependency $\bowtie\mathbf{S}$ whenever $\mathbf{S}$ is a connected subset of $\mathbf{R}$.

If $\mathbf{r}$ is a database over $\mathbf{R}$, if $\mathbf{S} \subseteq \mathbf{R}$, if $\mathbf{s} \subseteq \mathbf{r}$, and if $\mathbf{s}$ is a database over $\mathbf{S}$, then we say that $\mathbf{s}$ is the *subdatabase* over $\mathbf{S}$. It is not hard to see that property (4) says that for every connected subset $\mathbf{S}$ of $\mathbf{R}$ and every consistent database $\mathbf{r}$ over $\mathbf{R}$, if $\mathbf{s}$ is the subdatabase over $\mathbf{S}$, then $\bowtie\mathbf{s}$ is a projection of $\bowtie\mathbf{r}$.

One of the motivations for this paper was the question of whether every α-acyclic hypergraph $\mathbf{R}$ enjoys property (4). The answer is "no," since there are α-acyclic hypergraphs that are not γ-acyclic, such as the hypergraph in Figure 13. This

EMP_WORK:

EMP	DEPT	SAL
Fagin	CS	$200K

DEPT_INFO:

DEPT	CITY	MGR
CS	San Jose	Peled

EMP_HOME:

EMP	STREET	CITY	CHILD
Fagin	162 Loma Alta	Los Gatos	Joshua

FIGURE 20

hypergraph is not γ-acyclic and so violates property (4). In the case of this hypergraph, the join of the {SUPPLIER, PROJECT, DATE} relation with the {PROJECT, PART, COUNT} relation might introduce a SUPPLIER, PART, PROJECT triple that does not appear in the SUPPLIER, PART, PROJECT relation (the "connection trap" [15]).

(5) *There is a unique relationship among each set of attributes, for each consistent database over* **R**. Let **r** be a consistent database over **R**. By a *relationship* among a set $X \subseteq \bigcup \mathbf{R}$ of attributes, we mean a relation $(r_{i_1} \bowtie \cdots \bowtie r_{i_k})[X]$, where $X \subseteq R_{i_1} \cup \cdots \cup R_{i_k}$ and $\{R_{i_1}, \ldots, R_{i_k}\}$ is connected. Thus, some of the "base" relations **r** are combined, as usual, by taking joins (where none of these joins are Cartesian products), and the result is projected onto X. Property (5) says that the resulting relation is unique. It is sometimes convenient to refer to a relationship among X as an X *relationship.* Atzeni and Parker [3] discuss the power of assuming a unique relationship among each set of attributes (they call this the *Relationship Uniqueness Assumption*). They and others (e.g., Sagiv [35]) note that this assumption is made commonly, either explicitly or implicitly, in many papers on database design.

Let us consider an example which is slightly more elaborate than the example in the introduction. Assume that the database scheme consists of three relation schemes: an EMP_WORK relation scheme with attributes EMP (for "employee"), DEPT (for department), and SAL (for "salary"); a DEPT_INFO relation scheme with attributes DEPT, CITY, and MGR; and a EMP_HOME relation scheme with attributes EMP, STREET, CITY, and CHILD. See Figure 20 for an example of one tuple in each relation. In this example, there are two distinct {EMP, CITY} relationships. One, which has the tuple (Fagin, San Jose), relates an employee to the city where he works. The other, which has the tuple (Fagin, Los Gatos), relates an employee to the city where he lives. The database scheme is γ-cyclic (it is even α-cyclic).

However, assume that we were to rename the attribute CITY in the DEPT_INFO relation scheme to be WORK_CITY, and the attribute CITY in the EMP_HOME relation scheme to be HOME_CITY (see Figure 21). There is now a unique {EMP, WORK_CITY} relationship, which includes the tuple (Fagin, San Jose), and a unique {EMP, HOME_CITY} relationship, which includes the tuple (Fagin, Los Gatos). The database scheme of Figure 21 is γ-acyclic.

Knowing that relationships are unique make it possible to greatly simplify the form of queries. Thus, the simplest SQL [13] query to find all EMPs associated with

EMP_WORK:

EMP	DEPT	SAL
Fagin	CS	$200K

DEPT_INFO:

DEPT	WORK_CITY	MGR
CS	San Jose	Peled

EMP_HOME:

EMP	STREET	HOME_CITY	CHILD
Fagin	162 Loma Alta	Los Gatos	Joshua

FIGURE 21

the WORK__CITY San Jose for the database scheme of Figure 21 is

SELECT EMP
FROM EMP__WORK, DEPT__INFO
WHERE EMP__WORK.DEPT = DEPT__INFO.DEPT
AND DEPT__INFO.WORK__CITY = 'San Jose.'

However, by property (5), it is possible instead to unambiguously pose the query

SELECT EMP WHERE WORK__CITY = 'San Jose.' (8.1)

The result is obtained by finding the unique {EMP, WORK__CITY} relationship and then selecting out those tuples where the CITY entry is 'San Jose.' The desirability of being able to pose queries such as (8.1), with such a simple syntax, has been discussed by Ullman [37]. Not only is the query (8.1) easier to pose and simpler to understand than the SQL query, but also the system has a great deal of flexibility in optimizing how to find the result of the query. The system's choice of which relations to join (if there are several possibilities) might depend, for example, on which indices are present. The system might be able to exploit the fact that whatever relations in the database are joined together, the join (i.e., the join expression, as defined in Section 7) is guaranteed to be monotone, and so, efficient. For, we are only allowing joins over connected subsets **S** of **R**, which are themselves connected, γ-acyclic hypergraphs, since **R** is; and, because **S** is γ-acyclic, it follows from Theorem 8.1 below that every connected join expression over **S** is monotone. (However, when we project the result of the join onto the desired attributes, the number of tuples might, of course, decrease.)

Languages such as SQL are considered "high-level," since it is not necessary to state the access paths (such as which indices to utilize) explicitly. Similarly, we have seen that in a γ-acyclic database scheme, it is possible to make use of a still higher-level language, in which it is not even necessary to specify which relations must be joined to obtain the answer the user desires.

Aho and Kernighan [2] have developed a query system called "*q*." Given a set X of attributes, *q* searches through a "rel file" to determine which relations to join to find the X relationship. If the database scheme obeys property (5), that is, if it is γ-acyclic, then a rel file is unnecessary.

(6) **R** *has a loop-free Bachman diagram.* If **R** is a hypergraph, then we define *Bachman*(**R**) to be the hypergraph obtained by closing **R** under intersection. Thus, a set S is in Bachman(**R**) if and only if either $S \in \mathbf{R}$ or S is the intersection of two or more members of **R**. We note that both Lien [31] and Yannakakis [38] include also in Bachman(**R**) all singleton edges $\{A\}$, for each node A. We do not do so, since (as noted in [31]), this is really unnecessary. We leave to the reader the exercise of showing that Bachman(**R**) is γ-acyclic if and only if **R** is.

For our purposes it is convenient to define the *Bachman diagram* of **R** [5] to be an undirected graph, with nodes the members of Bachman(**R**), and with an edge between two nodes S and T of Bachman(**R**) iff (i) $S \subsetneq T$, and (ii) there is no W in Bachman(**R**) such that $S \subsetneq W \subsetneq T$. (The usual definition has a direction on these edges and thus yields a *directed* graph.) A *loop-free Bachman diagram* [31, 38] is a Bachman diagram that is a tree. If Bachman(**R**) is loop-free, then we say that **R** *has a loop-free Bachman diagram.* Yannakakis [38] discusses various properties of loop-free Bachman diagrams, and in particular shows the equivalence of properties (4) and (6).

(7) **R** *has a unique minimal connection among each set X of nodes.* Assume that the user wishes to obtain the projection onto X of the union of all lossless joins that involve (among others) attributes X. For motivation as to why a user would wish to obtain such a union, the reader is referred to [33, 37, 38].

If the database is consistent, then every lossless join (projected onto X) gives the same answer, and so it is easy to take such a union. If we do not assume consistency, then in general it might be quite an undertaking computationally to obtain such a union. We now describe a situation where the union can be obtained via a single lossless join, even if the database is not consistent.

In γ-acyclic schemes, lossless joins correspond to connected joins (see property (4)). Therefore we shall discuss connected, rather than lossless, joins.

Instead of assuming that the database is consistent, we shall make a weaker assumption, which we call the *subset condition.* The subset condition says that whenever R_1 and R_2 are relation schemes in the database scheme and $R_1 \subseteq R_2$, then $r_2[R_1] \subseteq r_1$, where r_i is the R_i relation in the database ($i = 1, 2$). Yannakakis [38] calls the individual assumptions in the subset condition *existence constraints.* Existence constraints are special cases of *inclusion dependencies* [12]. We note that Codd [17] assumed existence constraints involving his E-relations.

Let X be a subset of the nodes of **R**, and let **V** be a connected set of k distinct members $V_1, \ldots, V_k$ of Bachman(**R**). Following Yannakakis [39], we say that **V** is a *unique minimal connection (among members of X,* or simply, *among X)* if (i) $X \subseteq V_1 \cup \cdots \cup V_k$, and (ii) whenever $\mathbf{W} = \{W_1, \ldots, W_p\}$ is a connected subset of Bachman(**R**) with $X \subseteq W_1 \cup \cdots \cup W_p$, then there are k distinct members $W_{i_1}, \ldots, W_{i_k}$ of **W** (where k is the cardinality of **V**) such that $V_j \subseteq W_{i_j}$, for $1 \leq j \leq k$.

Yannakakis [39] observed that if **R** has a loop-free Bachman diagram (property (6) above), then this set **V** can be obtained by simply taking the maximal members of the smallest connected subgraph of the Bachman diagram of **R** that contains X. In other words, let $R_{i_1}, \ldots, R_{i_q}$ be a minimal (i.e., q is as small as possible) set of nodes of Bachman(**R**) such that $X \subseteq (R_{i_1} \cup \cdots \cup R_{i_q})$, and let **V** contain R_{i_j} ($1 \leq j \leq q$) precisely if there is no p ($1 \leq p \leq q$) such that $R_{i_j} \subsetneq R_{i_p}$.

We now mention an application of the unique minimal connection, which is of important practical use when the database is not necessarily consistent. This application was noted by Yannakakis [38]. (Yannakakis worked in the context of *weak* or *containing instances* [28], but the results are equivalent to what we shall state below.)

Assume now that the user requests an X relationship in the database, where X is a set of attributes. At least in principle, the response of the system is as follows (we neglect the issue of optimization, and describe the result in operational terms):

(1) For each relation scheme S in Bachman(**R**) but not in **R**, the system forms a new relation s over S by letting s be $\bigcup \{r[S]: r \in \mathbf{r}$ and **S** is a subset of the relation scheme of $r\}$. (Here **r** is the database over **R**.) The result is a new database **s** over Bachman(**R**), which contains all of the relations in the original database **r**, along with new relations over relation schemes in Bachman(**R**) − **R**. It is easy to see that since the original database **r** obeys the subset condition, so does the new database **s** over Bachman(**R**).
(2) If **V** is the unique minimal connection among X, then the response of the system to the user's query is $(\bowtie \mathbf{v})[X]$, where **v** is the subdatabase, over **V**, of **s**.

Let us denote by v this result $(\bowtie \mathbf{v})[X]$. Let $w = (\bowtie \mathbf{w})[X]$ be another X relationship. That is, (a) **W** is a connected subset of Bachman(**R**), (b) $X \subseteq \bigcup \mathbf{W}$, and (c) **w** is the subdatabase over **W**. It follows easily from the definition of unique minimal connection that $w \subseteq v$.

Thus, not only does the system answer the query by taking a connected join, but furthermore, this result contains every tuple that can be obtained by taking any connected join (which contains the desired attributes). The philosophy is that this response is probably what the user intends. If the user wants something different, then he can explicitly spell out what he wants. Thus, in the usual case, the user can specify what he wants in a high-level manner, and the system gives him a meaningful response, which should correspond exactly to what he desires a large proportion of the time. For a more extensive discussion of this philosophy, see [33, 37, 38].

Maier and Ullman [32] demonstrate another sense in which there is a unique connection among each set of nodes in an α-acyclic hypergraph. Yannakakis' notion of unique minimal connection is not only stronger, but, we believe, more natural.

PROOF OF EQUIVALENCE. We now show that the properties (1)–(7) described above are equivalent.

THEOREM 8.1. *Let* **R** *be a connected hypergraph. The following are equivalent*:

(1) **R** *is* γ*-acyclic.*
(2) *Every connected join expression over* **R** *is monotone.*
(3) *Every connected, sequential join expression over* **R** *is monotone.*
(4) *The join dependency* $\bowtie$**R** *implies that every connected subset of* **R** *has a lossless join.*
(5) *There is a unique relationship among each set of attributes for each consistent database over* **R**.
(6) **R** *has a loop-free Bachman diagram.*
(7) **R** *has a unique minimal connection among each set X of nodes.*

PROOF. It is convenient for us to introduce two new properties, which we shall call properties (2′) and (4′). We shall prove that (1) ⇒ (3) ⇒ (4′) ⇒ (4) ⇒ (2′) ⇒ (1), that (4′) ⇒ (2) ⇒ (2′), and that (4) ⇒ (5) ⇒ (4). Yannakakis shows that (4) and (6) are equivalent [38] and that (6) and (7) are equivalent [39] (we shall not show these equivalences). Taken together, these implications give us Theorem 8.1.

It is an instructive exercise (left to the reader) to prove directly the equivalence of (1) and (6). A helpful lemma is the fact (noted above) that **R** is γ-acyclic if and only if Bachman(**R**) is γ-acyclic. It is also helpful to make use of Definition 3 of γ-cyclicity.

We now define properties (2′) and (4′).

(2′) Let θ be a connected join expression over $\mathbf{R}$, let $\mathbf{r}$ be a consistent database over $\mathbf{R}$, and let $(\theta_1 \bowtie \theta_2)$ be a subexpression of θ. Then $\theta_1(\mathbf{r})$ and $\theta_2(\mathbf{r})$ are consistent.

If we replace "consistent database" in (2′) by "pairwise consistent database," then it is not hard to see that the result is exactly what (2) says. In particular, (2) $\Rightarrow$ (2′), since every consistent database is pairwise consistent.

(4′) Assume that $\mathbf{S} \subseteq \mathbf{R}$ is connected, that $\mathbf{r}$ is a pairwise consistent database over $\mathbf{R}$, and that $\mathbf{s}$ is the subdatabase over $\mathbf{S}$. Then $\bowtie\mathbf{s}$ is a projection of $\bowtie\mathbf{r}$.

If we replace "pairwise consistent database" in (4′) by "consistent database," then it is not hard to see that the result is exactly what (4) says. In particular, (4′) $\Rightarrow$ (4), since every consistent database is pairwise consistent.

(1) $\Rightarrow$ (3): Assume that (3) is false. We shall show that (1) is false, that is, that $\mathbf{R}$ is γ-cyclic. Since (3) is false, there is an ordering $R_1, \ldots, R_n$ of $\mathbf{R} = \{R_1, \ldots, R_n\}$, a pairwise consistent database $\mathbf{r} = \{r_1, \ldots, r_n\}$ over $\mathbf{R}$, and an integer j $(1 \leq j < n)$ such that

(a) $\{R_1, \ldots, R_i\}$ is connected for each i $(1 \leq i \leq n)$, and
(b) $r_1 \bowtie \cdots \bowtie r_j$ is not consistent with r_{j+1}.

We assume that j is minimal, so that (b) holds. Thus $(r_1 \bowtie \cdots \bowtie r_i)$ is consistent with r_{i+1}, if $i < j$.

Denote $(R_1 \cup \cdots \cup R_j) \cap R_{j+1}$ by S. Thus S is the set of attributes that $r_1 \bowtie \cdots \bowtie r_j$ has in common with r_{j+1}. We know that $(r_1 \bowtie \cdots \bowtie r_j)[S] \neq r_{j+1}[S]$, since $r_1 \bowtie \cdots \bowtie r_j$ is not consistent with r_{j+1}.

Select k $(1 \leq k \leq j)$ so that $R_k \cap S$ is as big as possible (i.e., has as many nodes as possible). Thus $R_i \cap S$ is no bigger than $R_k \cap S$ if $1 \leq i \leq j$. We now show that $S \not\subseteq R_k$. For, assume that $S \subseteq R_k$; we shall derive a contradiction. Since $\mathbf{r}$ is pairwise consistent, it follows that r_k and r_{j+1} are consistent. Thus $r_k[S] = r_{j+1}[S]$. By our minimality assumption on j, we know that $r_1 \bowtie \cdots \bowtie r_i$ is consistent with r_{i+1}, if $1 \leq i < j$. Thus no tuples are lost in taking the sequence following of joins:

$$
\begin{aligned}
&r_1 \bowtie r_2,\\
&(r_1 \bowtie r_2) \bowtie r_3,\\
&\quad\vdots\\
&(\cdots((r_1 \bowtie r_2) \bowtie r_3) \cdots \bowtie r_j).
\end{aligned}
$$

Since $k \leq j$, it follows that r_k is consistent with $r_1 \bowtie \cdots \bowtie r_j$. Thus $r_k[S] = (r_1 \bowtie \cdots \bowtie r_j)[S]$. Since also $r_k[S] = r_{j+1}[S]$, we have $(r_1 \bowtie \cdots \bowtie r_j)[S] = r_{j+1}[S]$. This is a contradiction. So, $S \not\subseteq R_k$.

Since $S \not\subseteq R_k$, there is a node v_1 in $S - R_k$. Since $\{R_1, \ldots, R_j\}$ is a connected set of edges, since R_k is an edge in this set, and since v_1 is a node that appears in this set, there are $S_1, \ldots, S_p$ such that

(i) $S_1 = R_k$,
(ii) $S_i \cap S_{i+1} \neq \emptyset$, for $1 \leq i < p$,
(iii) each S_i is one of $R_1, \ldots, R_j$ $(1 \leq i \leq p)$,
(iv) $v_1 \in S_p$, and
(v) p is as small as possible.

Thus $S_1, \ldots, S_p$ is the shortest path (within $\{R_1, \ldots, R_j\}$) from R_k to an edge containing v_1. In particular, $v_1 \notin S_i$, if $1 \leq i < p$. Note that $S_1, \ldots, S_p$ are distinct, since $S_1, \ldots, S_p$ is a shortest path.

Now $S_p \cap S$ has no more nodes than $R_k \cap S$, by maximality of $R_k \cap S$. Since $S_p \cap S$ contains v_1, which is not in $R_k \cap S$, it follows that $R_k \cap S$ contains a node v_2 not in $S_p \cap S$.

Find m ($1 \leq m < p$) as big as possible so that S_m contains v_2. There is such an m, since S_1 ($=R_k$) contains v_2. By (ii) above, we can find nodes x_i ($m \leq i < p$) such that $x_i \in S_i \cap S_{i+1}$. We now show that

$$(S_p, v_1, R_{j+1}, v_2, S_m, x_m, S_{m+1}, x_{m+1}, \ldots x_{p-1}, S_p) \tag{8.2}$$

is a weak γ-cycle. Note that v_1 and v_2 are both in R_{j+1}, since they are both in $S \subseteq R_{j+1}$. By construction, v_1 is in S_p but not in S_i, for $m \leq i < p$; similarly, v_2 is in S_m but not in S_i, for $m < i \leq p$. In particular, R_{j+1}, S_p, and S_m are all distinct, and so (8.2) contains at least three distinct edges. The nodes x_i ($m \leq i < p$) are distinct, or else the path $S_1, \ldots, S_p$ could have been shorter. Further, v_1 does not equal any x_i, since v_1 is not in S_i if $m \leq i < p$. Similarly, v_2 does not equal any x_i. Since we see also that $v_1 \neq v_2$ (because v_1 is in S_p and v_2 is not), this shows that all of the nodes v_1, v_2, x_m, $x_{m+1}, \ldots, x_{p-1}$ of (8.2) are distinct. Similarly, the edges of (8.2) are distinct. It follows from what we have just shown that (8.2) is indeed a weak γ-cycle. Thus $\mathbf{R}$ is γ-cyclic, which was to be shown.

(3) $\Rightarrow$ (4′): Assume (3). Assume that $\mathbf{S} \subseteq \mathbf{R}$ is connected, that $\mathbf{r}$ is a pairwise consistent database over $\mathbf{R}$, and that $\mathbf{s}$ is the subdatabase over $\mathbf{S}$. We must show that $\bowtie\mathbf{s}$ is a projection of $\bowtie\mathbf{r}$.

Since $\mathbf{S}$ and $\mathbf{R}$ are each connected, there is an ordering $R_1, \ldots, R_n$ of $\mathbf{R}$ such that

(a) $\mathbf{S} = \{R_1, \ldots, R_m\}$, where m is the cardinality of $\mathbf{S}$ (thus the members of $\mathbf{S}$ form an "initial segment" of the ordering $R_1, \ldots, R_n$); and
(b) $\{R_1, \ldots, R_i\}$ is connected for each i ($1 \leq i \leq n$).

Then $(\cdots ((R_1 \bowtie R_2) \bowtie R_3) \cdots \bowtie R_n)$ is a connected, sequential join expression over $\mathbf{R}$. By (3), it is monotone. It follows easily that $r_1 \bowtie \cdots \bowtie r_i$ is a projection of $r_1 \bowtie \cdots \bowtie r_n$ for each i, and, in particular, for $i = m$. Thus, $\bowtie\mathbf{s}$ is a projection of $\bowtie\mathbf{r}$, which was to be shown.

(4′) $\Rightarrow$ (4): Already shown, in our comments after the definition of property (4′).

(4) $\Rightarrow$ (2′): Assume (4). Let θ be a connected join expression over $\mathbf{R}$, let $\mathbf{r}$ be a consistent database over $\mathbf{R}$, and let $(\theta_1 \bowtie \theta_2)$ be a subexpression of θ. To prove (2′), we must show that $\theta_1(\mathbf{r})$ and $\theta_2(\mathbf{r})$ are consistent.

Assume that θ_1 is over $\mathbf{S} \subseteq \mathbf{R}$ and that θ_2 is over $\mathbf{T} \subseteq \mathbf{R}$. By connectedness of θ we know that $\mathbf{S}$ and $\mathbf{T}$ are each connected. Let $\mathbf{s}$ (respectively, $\mathbf{t}$) be the subdatabase over $\mathbf{S}$ (respectively, $\mathbf{T}$). By (4), each of $\bowtie\mathbf{s}$ and $\bowtie\mathbf{t}$ are projections of $\bowtie\mathbf{r}$. Hence, $\bowtie\mathbf{s}$ and $\bowtie\mathbf{t}$ are consistent. Now $\theta_1(\mathbf{r}) = \bowtie\mathbf{s}$, and $\theta_2(\mathbf{r}) = \bowtie\mathbf{t}$. So $\theta_1(\mathbf{r})$ and $\theta_2(\mathbf{r})$ are consistent. This was to be shown.

(2′) $\Rightarrow$ (1): Assume that (1) is false, that is, that $\mathbf{R} = \{R_1, \ldots, R_n\}$ is γ-cyclic. We shall show that (2′) is false.

Since $\mathbf{R}$ is γ-cyclic, we know by Definition 2 of γ-cyclicity that $\mathbf{R}$ has a weak

γ-cycle $(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1})$. Define

$$\begin{aligned} \mathbf{E} &= \{S_3, S_4, \ldots, S_m, S_1\}, \\ E_1 &= S_1, \\ E_2 &= S_3, \\ F &= S_2, \\ A_1 &= x_1, \\ A_2 &= x_2. \end{aligned}$$

It is easy to see that $\mathbf{E}$ is a connected set of edges, that E_1 and E_2 are distinct edges in $\mathbf{E}$, that F is an edge not in $\mathbf{E}$, and that A_1 and A_2 are distinct nodes such that

(i) A_1 is in E_1 but in no other edge of $\mathbf{E}$,
(ii) A_2 is in E_2 but in no other edge of $\mathbf{E}$, and
(iii) A_1 and A_2 are in F.

Let r be a relation with attributes $R_1 \cup \cdots \cup R_n$, and with exactly two tuples. The first tuple has all 0's, and the second tuple has all 0's except in the A_1 and A_2 entries, where it has 1's. Let $\mathbf{r} = \{r_1, \ldots, r_n\}$ be a database over $\mathbf{R}$, where $r_i = r[R_i]$ for each i $(1 \leq i \leq n)$. So, $\mathbf{r}$ is consistent.

Assume that the distinct members of $\mathbf{E}$ are $E_1, \ldots, E_m$. Let $\mathbf{e} = \{e_1, \ldots e_m\}$ be the subdatabase of $\mathbf{r}$ over $\mathbf{E}$, and let f be the member of $\mathbf{r}$ that is over F.

Clearly, $e_1 \bowtie e_2$ is a relation with exactly four tuples: the (A_1, A_2) entries of the four tuples are, respectively, (0, 0), (0, 1), (1, 0), and (1, 1). The remaining entries of all four tuples are all 0's. Also, $e_3 \bowtie \cdots \bowtie e_m$ is a one-tuple relation, where the one tuple has all 0's (recall that $E_3 \cup \cdots \cup E_m$ does not contain either A_1 or A_2). Thus, $e_1 \bowtie e_2 \bowtie \cdots \bowtie e_m$ has exactly four tuples, where the (A_1, A_2) entries are as before and the other entries are all 0's. However, there are only two (A_1, A_2) entries in f, namely, (0, 0) and (1, 1). So $e_1 \bowtie \cdots \bowtie e_m$ is not consistent with f.

Since $\{E_1, \ldots, E_m\}$, $\{E_1, \ldots, E_m, F\}$, and $\mathbf{R}$ are each connected, there is an ordering $R_1, \ldots, R_n$ of $\mathbf{R}$ such that

(a) $\mathbf{E} = \{R_1, \ldots, R_m\}$ (i.e., $\mathbf{E}$ forms an initial segment),
(b) $F = R_{m+1}$ (that is, F is next after $\mathbf{E}$), and
(c) $\{R_1, \ldots, R_i\}$ is connected for each i $(1 \leq i \leq n)$.

For each k, with $1 \leq k \leq n$, define the join expression θ_k to be $(\cdots ((R_1 \bowtie R_2) \bowtie R_3) \cdots \bowtie R_k)$. Then θ_n is a connected join expression over $\mathbf{R}$. Also, $(\theta_m \bowtie R_{m+1})$ is a subexpression of θ_n. However, if $\mathbf{r}$ is the consistent database described earlier, then $\theta_m(\mathbf{r})$ is not consistent with $R_{m+1}(\mathbf{r})$, that is, $(r_1 \bowtie \cdots \bowtie r_m)$ is not consistent with r_{m+1}, since $r_1 \bowtie \cdots \bowtie r_m = e_1 \bowtie \cdots \bowtie e_m$, and $r_{m+1} = f$. Thus (2′) is false, which was to be shown.

(4′) $\Rightarrow$ (2): This is the same as the proof that (4) $\Rightarrow$ (2′), except that "pairwise consistent" is replaced by "consistent."

(2) $\Rightarrow$ (2′): Already shown, in our comments after the definition of property (2′).

(4) $\Rightarrow$ (5): Assume (4); we shall show (5). Let $\mathbf{r}$ be a consistent database over $\mathbf{R}$. Let $\mathbf{S} = \{S_1, \ldots, S_p\}$ and $\mathbf{T} = \{T_1, \ldots, T_q\}$ be connected subsets of $\mathbf{R}$. Assume that $X \subseteq \cup\mathbf{S}$ and $X \subseteq \cup\mathbf{T}$. Let $\mathbf{s}$ (respectively, $\mathbf{t}$) be the subdatabase of $\mathbf{r}$ over $\mathbf{S}$ (respectively, $\mathbf{T}$). By (4), we know that $\bowtie\mathbf{s}$ is a projection of $\bowtie\mathbf{r}$. Hence, $(\bowtie\mathbf{s})[X] = (\bowtie\mathbf{r})[X]$. Similarly, $(\bowtie\mathbf{t})[X] = (\bowtie\mathbf{r})[X]$. Hence, $(\bowtie\mathbf{s})[X] = (\bowtie\mathbf{t})[X]$. This was to be shown.

(5) $\Rightarrow$ (4): Assume (5); we shall show (4). Let $\mathbf{r}$ be a consistent database over $\mathbf{R}$, let $\mathbf{S}$ be a connected subset of $\mathbf{R}$, and let $\mathbf{s}$ be the subdatabase of $\mathbf{r}$ over $\mathbf{S}$. Let $X = \cup \mathbf{S}$. By (5), we know that $(\bowtie \mathbf{s})[X] = (\bowtie \mathbf{r})[X]$, that is, $\bowtie \mathbf{s} = (\bowtie \mathbf{r})[X]$. Hence, $\bowtie \mathbf{s}$ is a projection of $\bowtie \mathbf{r}$. This proves (4). □

9. *Polynomial-Time Algorithms for Determining Degree of Acyclicity*

We now show that there are polynomial-time algorithms for determining whether a hypergraph is Berge-acyclic, α-acyclic, β-acyclic, and γ-acyclic.

In the algorithms we now describe, we make no attempt at optimal efficiency, since we are concerned here only with the question of polynomial-time recognition. It is an interesting problem to find more efficient recognition algorithms.

9.1 BERGE-ACYCLICITY. It is easy to see that the usual breadth-first search algorithm for determining acyclicity of an ordinary undirected graph (in which we start with an edge and propagate the graph outward while watching to see if it "folds back on itself" by touching a previously used node) generalizes neatly and easily to determining Berge-acyclicity. The simple details are left to the reader.

9.2 α-ACYCLICITY. Beeri et al. [8] prove that the following simple algorithm, called *Graham's algorithm* [26, 40], is a test for α-acyclicity. The algorithm applies the following two rules to $\mathbf{R} = \{R_1, \ldots, R_n\}$ repeatedly until neither can be applied:

(a) If A is an attribute that appears in exactly one R_i, then delete A from R_i.
(b) Delete one R_i if there is an R_j with $j \neq i$ such that $R_i \subseteq R_j$.

Intuitively, rules of type (a) remove attributes that cannot have any effect on α-cyclicity or α-acyclicity, and rules of type (b) causes a hypergraph to be replaced by its reduction.

If the algorithm terminates with the empty set, then the hypergraph is α-acyclic; otherwise, the hypergraph is α-cyclic. We note that it is not hard to show that the algorithm is Church–Rosser. That is, the set that the algorithm terminates with is independent of the sequence of steps taken in executing the algorithm and depends only on the input.

Example 9.1. Let us apply Graham's algorithm to the hypergraph of Figure 3, with edges ABC, CDE, EFA, and ACE. Nodes B, D, and F each appear in only one edge, and so they are each deleted by applications of rule (a) of Graham's algorithm. We are then left with edges AC, CE, EA, and ACE. Now edge AC is a subset of edge ACE, so by an application of rule (b) of the algorithm, this edge is deleted. This leaves us with edges CE, EA, and ACE. Similarly, edges CE and EA are deleted by applications of rule (b). We are then left with only one edge, namely ACE. Each of the nodes A, C, and E now appear in only one edge, and so by applications of rule (a), they are each deleted. We are left with the empty set, and so the hypergraph is α-acyclic. □

It is obvious that Graham's algorithm is a polynomial-time algorithm. Tarjan and Yannakakis [36] have recently obtained a linear-time algorithm for determining α-acyclicity.

9.3 β-ACYCLICITY. We shall base our polynomial-time algorithm on Definition 1 of β-acyclicity; that is, we shall determine whether or not there is a β-cycle.

If $(S_1, \ldots, S_m, S_{m+1})$ is a β-cycle (respectively, pure cycle), then we say that (S_1, S_2, S_3) *begins* the β-cycle (respectively, pure cycle).

We now give a polynomial-time algorithm for determining whether $\mathscr{S} = (S_1, S_2, S_3)$ begins some β-cycle of $\mathbf{R}$, if S_1, S_2, and S_3 are distinct edges in $\mathbf{R}$. Let $X = S_1 \cap S_2 \cap S_3$, and let $S_i' = S_i - X$, for $i = 1, 2, 3$. If either $S_1' \cap S_2'$ or $S_2' \cap S_3'$ is empty, then $\mathscr{S}$ does not begin any β-cycle of $\mathbf{R}$. Therefore, assume that $S_1' \cap S_2'$ and $S_2' \cap S_3'$ are both nonempty.

Let $\mathbf{T} = \{E \in \mathbf{R}: (E = S_1)$ or $(E = S_3)$ or $(X \subsetneq E$ and $E \cap S_2' = \emptyset)\}$. Note in particular that $S_2 \notin \mathbf{T}$. Let $\mathbf{T}' = \{E - X: E \in \mathbf{T}\}$. In particular, S_1' and S_3' are in $\mathbf{T}'$. We now show that S_1' and S_3' are in the same connected component of $\mathbf{T}'$ if and only if $\mathscr{S}$ begins a β-cycle of $\mathbf{R}$.

Assume first that $\mathscr{S}$ begins a β-cycle $(S_1, S_2, S_3, \ldots, S_{m+1})$ of $\mathbf{R}$ (where, of course, $S_{m+1} = S_1$). Then it is easy to see that $S_1 \cap \cdots \cap S_m = S_1 \cap S_2 \cap S_3$, that is, $S_1 \cap \cdots \cap S_m = X$. It is clear that S_1' and S_3' are then in the same connected component of $\mathbf{T}'$. Conversely, assume that S_1' and S_3' are in the same component of $\mathbf{T}'$. Find $E_1', \ldots, E_k'$ in $\mathbf{T}'$ such that

(i) $E_1' = S_3'$,
(ii) $E_k' = S_1'$,
(iii) $E_i' \cap E_{i+1}' \neq \emptyset$, and
(iv) k is as small as possible.

It is then easy to see that $(S_1', S_2', S_3', E_2', E_3', \ldots, E_k')$ is a pure cycle (in particular, by construction of $\mathbf{T}'$, we know that $E_i' \cap S_2' = \emptyset$, for $2 \leq i < k$). Define $E_i = E_i' \cup X$, for $1 \leq i \leq k$. By construction of $\mathbf{T}'$ we know that each E_i is an edge in $\mathbf{R}$. So $(S_1, S_2, S_3, E_2, E_3, \ldots, E_k)$ is a β-cycle.

There is a polynomial-time algorithm for determining connected components of a hypergraph (such as $\mathbf{T}'$). The algorithm is the obvious generalization of the usual algorithm in the case of ordinary undirected graphs for determining connected components. So, there is a polynomial-time algorithm for determining whether $\mathscr{S}$ begins a β-cycle.

Our polynomial-time algorithm for determining β-acyclicity goes as follows. Systematically cycle through all triples $\mathscr{S} = (S_1, S_2, S_3)$ of three distinct edges of $\mathbf{R}$ to see if at least one such $\mathscr{S}$ begins a β-cycle. If so, then $\mathbf{R}$ is β-cyclic; otherwise, $\mathbf{R}$ is β-acyclic.

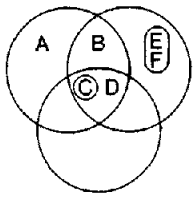
Graham [27] states that he has found a polynomial-time algorithm for determining whether a hypergraph has a Graham cycle. Thus, by the equivalence of Definition 4 of β-cyclicity with the other definitions, this gives another polynomial-time algorithm for determining β-acyclicity.

9.4 γ-ACYCLICITY. The following algorithm for testing γ-acyclicity is due to D'Atri and Moscarini [18]. It is similar in spirit to Graham's algorithm for determining α-cyclicity.

Apply the following rules repeatedly, in any order, until none can be applied:

(a) If a node is isolated (i.e., if it belongs to precisely one edge), then delete that node.
(b) If an edge is a singleton (i.e., if it contains exactly one node), then delete that edge (but do not delete the node from other edges that might contain it).
(c) If an edge is empty, then delete it.
(d) If two edges contain precisely the same nodes, then delete one of these edges.
(e) If two nodes are edge-equivalent, then delete one of them from every edge that contains it. (Recall that two nodes are *edge-equivalent* if they are in precisely the same edges.)

FIGURE 22



The algorithm clearly terminates. If the end result is the empty set of edges, then the original hypergraph is γ-acyclic; otherwise, it is γ-cyclic.

As in the case of Graham's algorithm, we note that it is not hard to show that this algorithm is Church–Rosser.

Remark. We shall often apply rule (d) implicitly, by simply dealing at all times with a *set* of edges (which has the effect of automatically removing duplicates). Also, it is natural to apply rule (c), the deletion of an empty edge, implicitly.

Example 9.2. Let us apply this algorithm to the hypergraph of Figure 22. The edges are

$$\begin{array}{cccccc} & B & C & D & E & F \\ A & B & C & D & & \\ & & C & & & \\ & & C & D & & \\ & & & & E & F \end{array}$$

(For convenience, we have put common vertices in the same column.) Node A is isolated, and edge $\{C\}$ is a singleton, so both are deleted, by rules (a) and (b). This leaves us with

$$\begin{array}{ccccc} B & C & D & E & F \\ B & C & D & & \\ & C & D & & \\ & & & E & F \end{array}$$

Nodes E and F are edge-equivalent, and so, by rule (e), we delete F from both edges that contain it. Similarly, nodes C and D are edge-equivalent, and so we delete D from all three edges that contain it. We are left with

$$\begin{array}{ccc} B & C & E \\ B & C & \\ & C & \\ & & E \end{array}$$

The third and fourth edges above are singletons, and so they are eliminated. This leaves

$$\begin{array}{ccc} B & C & E \\ B & C & \end{array}$$

Node E is isolated; after it is deleted, we are left with

$$B \quad C$$

$$B \quad C$$

These edges are identical, so we delete one by rule (d). We are left with

$$B \quad C$$

Both nodes are now isolated, and so they are deleted. We are left with a single empty edge, which is deleted by rule (c). The end result is the empty set of edges, and so the original hypergraph is γ-acyclic. □

THEOREM 9.3. *The algorithm just described correctly determines whether or not a hypergraph is γ-acyclic.*

PROOF. Assume first that the hypergraph is γ-cyclic. By Definition 1 we know that the hypergraph has a γ-cycle $(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1})$. It is easy to verify inductively on the number of steps that have been applied so far in running the algorithm (where a *step* consists of one application of a rule) that for each i $(1 \leq i \leq m)$, whenever a rule of the algorithm is applied, then either x_i or some node that is edge-equivalent to x_i at the time the rule is applied remains undeleted. In particular, after each step a γ-cycle of size at least m remains. Therefore, when the algorithm terminates, there is a γ-cycle of size at least m. Hence the algorithm does not terminate with the empty set, and so the algorithm correctly determines that the hypergraph is γ-cyclic.

Conversely, assume that the algorithm says that the hypergraph is γ-cyclic. We must show that the hypergraph is indeed γ-cyclic. Assume that the hypergraph is γ-acyclic; we shall derive a contradiction. Since the hypergraph (call it $\mathscr{H}$) is γ-acyclic, we know by Definition 1 that $\mathscr{H}$ has no γ-cycle. Let $\mathscr{H}'$ be the hypergraph that is the end result of applying the algorithm to the hypergraph $\mathscr{H}$. It is easy to see that when one of the rules in the algorithm is applied to a hypergraph with no γ-cycle, then the result is a hypergraph with no γ-cycle. It follows inductively (on the number of steps) that since $\mathscr{H}$ has no γ-cycle, neither does $\mathscr{H}'$. Thus $\mathscr{H}'$ is γ-acyclic. Since none of the rules in the algorithm can be applied to $\mathscr{H}'$, it follows that each edge of $\mathscr{H}'$ contains at least two nodes, each node is contained in at least two edges, and no two distinct nodes are edge-equivalent.

Let us say that a hypergraph is *nesting* if for each pair (E_1, E_2) of edges, either (a) $E_1 \subseteq E_2$, (b) $E_2 \subseteq E_1$, or (c) $E_1 \cap E_2 = \emptyset$. Thus every pair of edges is either comparable or disjoint. Let us call a hypergraph *intersecting* if it is not nesting. Thus a hypergraph is intersecting precisely if it has a pair of incomparable, nondisjoint edges.

We shall make use of the following simple fact several times.

FACT 1. *Let $\mathscr{I}$ be a nesting hypergraph, and let E be a minimal edge of $\mathscr{I}$ (i.e., there is no edge E' of $\mathscr{I}$ such that $E' \subsetneq E$). Then the nodes of E are all edge-equivalent.*

PROOF OF FACT 1. Let $\mathscr{I}$ be a nesting hypergraph, let E be a minimal edge of $\mathscr{I}$, and let x and y be distinct nodes of E. We must show that x and y are edge-equivalent. Assume not. Then there is an edge F that contains exactly one of x or y, say x. Since E is minimal, we know that $F \not\subseteq E$. Thus there is a node z in F but not E. Since also y is in E but not F, and since x is in $E \cap F$, it follows that E and F are incomparable and nondisjoint. This is a contradiction (since $\mathscr{I}$ is nesting), which proves Fact 1.

Let us say that a node in a hypergraph is *bad* if either (a) it is in exactly one edge, or (b) it is edge-equivalent to another node. If E_1 and E_2 are distinct edges, then let us say that the pair (E_1, E_2) is a *bad pair of edges* if there is a bad node in each of the set differences $E_1 - E_2$ and $E_2 - E_1$.

Let us say that a hypergraph is *nonsingular* if every edge has at least two nodes. We shall prove the following.

FACT 2. *Every γ-acyclic, intersecting, nonsingular hypergraph has a bad pair of edges.*

We now show that Fact 2 gives us a contradiction. As we showed, the hypergraph $\mathscr{H}'$ defined above is γ-acyclic and nonsingular and has no bad nodes. We now show that $\mathscr{H}'$ is intersecting. Assume not. Then $\mathscr{H}'$ is nesting. Let E be a minimal edge of $\mathscr{H}'$. Edge E (and every edge of $\mathscr{H}'$) has at least two nodes. Let x and y be distinct nodes of E. By Fact 1, we know that x and y are edge-equivalent. But $\mathscr{H}'$ has no pair of distinct edge-equivalent nodes. This contradiction shows that $\mathscr{H}'$ is intersecting. Since $\mathscr{H}'$ is γ-acyclic, intersecting, and nonsingular, it follows from Fact 2 that $\mathscr{H}'$ has a bad pair of edges, and so $\mathscr{H}'$ has a bad node. But $\mathscr{H}'$ has no bad node. This is the desired contradiction. Thus we need only prove Fact 2 to prove the theorem.

We shall prove Fact 2 by induction on the number of edges in the hypergraph. The base case (of hypergraphs with only one edge) is immediate, since no hypergraph with only one edge is intersecting. Assume that Fact 2 holds for hypergraphs with less than n edges, and let $\mathscr{J}$ be a hypergraph with n edges that is γ-acyclic, intersecting, and nonsingular. We must show that $\mathscr{J}$ has a bad pair of edges.

Since $\mathscr{J}$ is intersecting, it has a pair (E, F) of edges that are incomparable and nondisjoint. Find such a pair (E, F) such that $E \cap F$ is as small as possible. Thus, if E' and F' are incomparable and nondisjoint edges of $\mathscr{G}$, then $|E' \cap F'| \geq |E \cap F|$. (Here $|X|$ is the cardinality of set X.)

Since $\mathscr{J}$ is γ-acyclic, it follows from Definition 4 that in the hypergraph that results by removing $E \cap F$ from every edge, what is left of E is not connected to what is left of F. Let us write $E \cap F$ as Q. Let $\mathscr{G}$ be a hypergraph with the same nodes as $\mathscr{J}$ and whose edges are precisely those edges of $\mathscr{J}$ that are not subsets of Q. Note that E and F are each edges in $\mathscr{G}$, since they are incomparable and their intersection is Q. For $\mathscr{G}$, too, it is the case that in the hypergraph that results by removing Q from every edge, what is left of E is not connected to what is left of F. We can thus partition the edges of $\mathscr{G}$ into two disjoint sets $\mathscr{E}$ and $\mathscr{F}$ such that $E \in \mathscr{E}$ and $F \in \mathscr{F}$, and such that

$$\text{whenever} \quad E' \in \mathscr{E} \quad \text{and} \quad F' \in \mathscr{F}, \qquad \text{then} \quad E' \cap F' \subseteq Q. \tag{9.1}$$

Since we have several hypergraphs we are now dealing with (namely, $\mathscr{E}$, $\mathscr{F}$, $\mathscr{G}$, and $\mathscr{J}$), it is convenient for us to subscript the notion of "bad" with the hypergraph we are discussing. For example, if we say that x is a bad$_{\mathscr{E}}$ node, we mean that either (i) x is in exactly one edge of $\mathscr{E}$, or else (ii) x is edge-equivalent (with respect to $\mathscr{E}$) to another node of $\mathscr{E}$, that is, x is in precisely the same edges of $\mathscr{E}$ as another node of $\mathscr{E}$. Similarly, we can speak of a bad$_{\mathscr{E}}$ pair of edges, etc.

We now prove three simple facts, each of which we shall use several times.

FACT 3. *Each edge of $\mathscr{E}$ either contains Q or is disjoint from Q.*

PROOF OF FACT 3. Assume that Fact 3 were false. Let E' be an edge of $\mathscr{E}$ that neither contains Q nor is disjoint from Q. Now $Q \not\subseteq E'$, by assumption, and so $Q \not\subseteq E' \cap F$. If we put this together with the fact that $E' \cap F \subseteq Q$ (which we know by (9.1)), it follows that $E' \cap F$ is strictly smaller than $Q = E \cap F$. Also, E' and F are

nondisjoint, since by assumption E' and $Q = E \cap F$ are nondisjoint. Now $E' \in \mathscr{E} \subseteq \mathscr{G}$, and by definition of $\mathscr{G}$ no edge of $\mathscr{G}$ is contained in Q. Therefore $E' \not\subseteq Q$. Hence there is a node e in $E' - Q$. But $E' \cap F \subseteq Q$ by (9.1), and so $e \notin F$. Since $e \in E' - F$, we know that $E' \not\subseteq F$. Further, $F \not\subseteq E'$: for, if $F \subseteq E'$, then $F = E' \cap F \subseteq Q \subseteq E$, where the next-to-the-last inclusion follows from (9.1); however, by our choice of E and F, we know that $F \not\subseteq E$. We have shown in this paragraph that E' and F are incomparable and nondisjoint, and that $E' \cap F$ is strictly smaller than $E \cap F$. This contradicts our minimality assumption in the choice of (E, F). Therefore, Fact 3 is proved.

FACT 4. *Assume that node a is in exactly one edge of $\mathscr{E}$, and that $a \notin Q$. Then a is bad$_{\mathscr{J}}$.*

PROOF OF FACT 4. It is sufficient to show that a is in exactly one edge of $\mathscr{J}$. Let E' be the edge of $\mathscr{E}$ that contains a. Assume that a is in another edge I of $\mathscr{J}$ other than E'. By assumption, we know that $I \notin \mathscr{E}$. If $I \in \mathscr{F}$, then by (9.1) we know $a \in Q$, a contradiction. So $I \notin \mathscr{G}$. Therefore, $I \in \mathscr{J} - \mathscr{G}$. But then $a \in Q$ by definition of $\mathscr{G}$. This contradiction completes the proof of Fact 4.

FACT 5. *Assume that a and b are nodes in $\mathscr{E}$ that are edge-equivalent with respect to $\mathscr{E}$. Assume also that neither a nor b is in Q. Then a is bad$_{\mathscr{J}}$.*

PROOF OF FACT 5. Let I be an edge of $\mathscr{J}$ that contains node a. Then $I \in \mathscr{G}$, since otherwise $a \in I \subseteq Q$, a contradiction. We now show that $I \in \mathscr{E}$. For if not, then $I \in \mathscr{G} - \mathscr{E} = \mathscr{F}$, so a is a node in both $\mathscr{E}$ and $\mathscr{F}$, and so by (9.1) it follows that $a \in Q$, a contradiction. We have shown that each edge of $\mathscr{J}$ that contains node a is an edge in $\mathscr{E}$. Similarly, the same is true about node b. Since a and b are edge-equivalent with respect to $\mathscr{E}$, it then follows immediately that a and b are edge-equivalent with respect to $\mathscr{J}$. Thus a is bad$_{\mathscr{J}}$. This completes the proof of Fact 5.

Now that Facts 3–5 are proved, we return to the proof of Fact 2 (which will complete the proof of the theorem).

We shall show that there is a bad$_{\mathscr{J}}$ node e which is an edge E_1 of $\mathscr{E}$ but which is not in Q. Identically, it follows that there is a bad$_{\mathscr{J}}$ node f which is in an edge F_1 of $\mathscr{F}$ but which is not in Q. From (9.1), we see that $E_1 \cap F_1 \subseteq Q$. Since $e \in E_1$ and $e \notin Q$, it follows that $e \notin F_1$. Thus $e \in E_1 - F_1$. Similarly, $f \in F_1 - E_1$. So (E_1, F_1) is a bad$_{\mathscr{J}}$ pair of edges. Hence, there is a bad$_{\mathscr{J}}$ pair of edges, which is exactly what we wished to show to complete the proof.

Thus, we need only show that $\mathscr{E}$ contains an edge E_1 that contains a bad$_{\mathscr{J}}$ node e where $e \notin Q$. There are two cases.

Case 1. $\mathscr{E}$ is nesting. There are two subcases.

Case 1a. There is an edge of $\mathscr{E}$ that is disjoint from F. Let G be a minimal edge of $\mathscr{E}$. Since some edge of $\mathscr{E}$ is disjoint from F, and since $\mathscr{E}$ is nesting, it is clear that G is disjoint from F. Let a and b be two distinct nodes of G. By Fact 1, nodes a and b are edge-equivalent with respect to $\mathscr{E}$. Since G is disjoint from F, it follows that neither a nor b is in Q (because $Q \subseteq F$). By Fact 5, a is bad$_{\mathscr{J}}$. Therefore, a is the desired bad$_{\mathscr{J}}$ node which is in an edge of $\mathscr{E}$ but not in Q.

Case 1b. No edge of $\mathscr{E}$ is disjoint from F. Let G be a maximal edge of $\mathscr{E}$. Then $Q \subsetneq E \subseteq G$ (where the last inclusion holds since G is maximal and $\mathscr{E}$ is nesting). Therefore, since G is maximal and $\mathscr{E}$ is nesting, it is clear that G contains a node e that is not in any other member of $\mathscr{E}$ and not in Q. By Fact 4, we know that e is bad$_{\mathscr{J}}$. Therefore, e is the desired bad$_{\mathscr{J}}$ node which is in an edge of $\mathscr{E}$ but not in Q.

Case 2. $\mathscr{E}$ is intersecting. Since $\mathscr{F}$ is γ-acyclic and nonsingular and $\mathscr{E} \subseteq \mathscr{F}$, it follows that $\mathscr{E}$ is γ-acyclic and nonsingular. Therefore, by our inductive assumption about Fact 2, we know that $\mathscr{E}$ has a $\text{bad}_{\mathscr{E}}$ pair (E_1, E_2) of edges. Let e_1 be a $\text{bad}_{\mathscr{E}}$ node in $E_1 - E_2$, and let e_2 be a $\text{bad}_{\mathscr{E}}$ node in $E_2 - E_1$. We now show that it is impossible for both e_1 and e_2 to be in Q. For, assume that e_1 and e_2 are both in Q. Since $e_1 \in E_1 \cap Q$, we know that E_1 is not disjoint from Q. So, by Fact 3, it follows that $Q \subseteq E_1$. Since $e_2 \in Q$, it follows that $e_2 \in E_1$, which is a contradiction. Therefore, one of e_1 or e_2, say e_1, is not in Q. Since e_1 is $\text{bad}_{\mathscr{E}}$, we know that either (i) e_1 is in exactly one edge of $\mathscr{E}$, or else (ii) e_1 is edge-equivalent (with respect to $\mathscr{E}$) to another node e_1' of $\mathscr{E}$. In case (i) it follows from Fact 4 that e_1 is the desired $\text{bad}_{\mathscr{F}}$ node which is in an edge of $\mathscr{E}$ but not in Q. So we can assume that case (ii) holds. If $e_1' \notin Q$, then it follows from Fact 5 that once again e_1 is the desired $\text{bad}_{\mathscr{F}}$ node which is in an edge of $\mathscr{E}$ but not in Q. Therefore, we can assume that $e_1' \in Q$.

Since $e_1 \notin E_2$ and since e_1 and e_1' are edge-equivalent (with respect to $\mathscr{E}$), it follows that $e_1' \notin E_2$. So, since $e_1' \in Q$, it follows that $Q \not\subseteq E_2$. Since $Q \not\subseteq E_2$, it follows from Fact 3 that Q is disjoint from E_2. Since e_2 is $\text{bad}_{\mathscr{E}}$, we know that either (i) e_2 is in exactly one edge of $\mathscr{E}$, or else (ii) e_2 is edge-equivalent (with respect to $\mathscr{E}$) to another node e_2' of $\mathscr{E}$. Now $e_2 \notin Q$, since Q is disjoint from E_2. Therefore, in case (i) it follows from Fact 4 that e_2 is the desired $\text{bad}_{\mathscr{F}}$ node which is in an edge of $\mathscr{E}$ but not in Q. So we can assume that case (ii) holds. Now $e_2' \notin Q$, since Q is disjoint from E_2. Therefore, it follows from Fact 5 that once again, e_2 is the desired $\text{bad}_{\mathscr{F}}$ node which is in an edge of $\mathscr{E}$ but not in Q. This completes the proof. □

We note that the above proof was inspired by the proof in [8] that Graham's algorithm recognizes precisely the α-acyclic hypergraphs.

The algorithm clearly runs in polynomial time. We remark that Yannakakis [38] shows that if $\mathbf{R}$ has a loop-free Bachman diagram, then $|\text{Bachman}(\mathbf{R})| \leq |\mathbf{R}| + 2|U|$, where $U = \cup\mathbf{R}$ is the set of all attributes. This provides another polynomial-time algorithm for determining γ-acyclicity, which we now describe. Start computing Bachman($\mathbf{R}$), but stop and announce γ-cyclicity if it becomes bigger than $|\mathbf{R}| + 2|U|$. Once Bachman($\mathbf{R}$) is computed (if we have not stopped and announced γ-cyclicity already), form the Bachman diagram of $\mathbf{R}$, and see if it is loop-free.

10. *Conclusions*

We have discussed the concepts of α-acyclicity, β-acyclicity, and γ-acyclicity for hypergraphs and for relational database schemes. These are all distinct, and each corresponds precisely to various desirable properties of relational database schemes. These concepts are also of interest from a graph-theoretic viewpoint, as natural generalizations of the notion of acyclicity from graphs to hypergraphs.

ACKNOWLEDGMENTS. The author is grateful to Shel Finkelstein and Jorma Rissanen for interesting discussions about Theorem 8.1(4) that led the author to consider the concept of γ-acyclicity. He is also grateful to Nat Goodman, Marc Graham, Dave Maier, Alberto Mendelzon, Oded Shmueli, Jeff Ullman, and Mihalis Yannakakis for helpful discussions. Special thanks go to Moshe Vardi for his careful reading of the paper and numerous suggestions, including a way to simplify the proof of Theorem 9.3.

REFERENCES

(Note. Reference [9] is not cited in the text.)

1. AHO, A.V., BEERI, C., AND ULLMAN, J.D. The theory of joins in relational databases. *ACM Trans. Database Syst. 4*, 3 (Sept. 1979), 297–314

2. AHO, A.V., AND KERNIGHAN, B.W. Private communication, Nov. 1981.
3 ATZENI, P., AND PARKER, D S., JR. Assumptions in relational database theory. In *Proc. 1st ACM SIGACT-SIGMOD Symp on Principles of Database Systems* (Los Angeles, Calif., Mar. 29–31, 1982), ACM, New York, 1982, pp. 1–9.
4. AUSIELLO, G , D'ATRI, A , AND MOSCARINI, M. Minimal coverings of acyclic database schemata. Proc. ONERA-CERT Toulouse Workshop on Logical Bases for Data Bases, Toulouse, France, 1982.
5 BACHMAN, C.W. Data structure diagrams. *Data Base 1,* 2 (1969), 4–10.
6 BATINI, C., D'ATRI, A., AND MOSCARINI, M Formal tools for top-down and bottom-up generation of acyclic relational schemata. Proc 7th Int. Conf. on Graph-Theoretic Concepts in Computer Science, Linz, Austria, 1981.
7. BEERI, C., FAGIN, R., MAIER, D., MENDELZON, A.O , ULLMAN, J D., AND YANNAKAKIS, M. Properties of acyclic database schemes. In *Proc. 13th Ann. ACM Symp. on Theory of Computing* (Milwaukee, Wisc., May 11–13, 1981), ACM, New York, 1981, pp. 355–362.
8. BEERI, C., FAGIN, R., MAIER, D , AND YANNAKAKIS, M. On the desirability of acyclic database schemes. *J. ACM 30,* 3 (July 1983), 479–513
9. BEERI, C , MENDELZON, A.O., SAGIV, Y., AND ULLMAN, J.D. Equivalence of relational database schemes *SIAM J Comput. 10,* 2 (June 1981), 352–370
10. BERGE, C. *Graphs and Hypergraphs.* North-Holland, New York, 1976.
11. BERNSTEIN, P.A., AND GOODMAN, N. The power of natural semijoins. *SIAM J. Comput. 10,* 4 (Nov. 1981), 751–771.
12. CASANOVA, M.A., FAGIN, R., AND PAPADIMITRIOU, C. Inclusion dependencies and their interaction with functional dependencies. In *Proc. ACM Symp on Principles of Database Systems* (Los Angeles, Calif., Mar. 29–31, 1982), ACM, New York, 1982, pp. 171–176
13 CHAMBERLIN, D D., ASTRAHAN, M.M., ESWARAN, K.P., GRIFFITHS, P.P., LORIE, R.A., MEHL, J.W., REISNER, P., AND WADE, B.W. SEQUEL 2: A unified approach to data definition, manipulation, and control. *IBM J Res. Dev. 20,* 6 (Nov. 1976), 560–575
14. CHASE, K. Join graphs and acyclic data base schemes. In *Proc. 7th Int. Conf. on Very Large Databases* (Cannes, France, Sept 9–11, 1981), ACM, New York, 1981, pp. 95–100.
15. CODD, E.F. Further normalization of the database relational model. In *Data Base Systems,* Courant Computer Science Symposia 6, R Rustin, Ed , Prentice-Hall, 1971, pp. 65–98.
16. CODD, E.F. Recent investigations into relational database systems. In *Proc. IFIP Congress 74,* North-Holland, New York, 1974, pp. 1017–1021.
17. CODD, E.F. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst. 4,* 4 (Dec. 1979), 397–434.
18. D'ATRI, A., AND MOSCARINI, M. Acyclic hypergraphs· Their recognition and top-down versus bottom-up generation. Tech. Rep. R.29, Consiglio Nazionale Delle Richerche, Istituto di Analisi dei Sistemi ed Informatica, 1982.
19. FAGIN, R. The decomposition versus the synthetic approach to relational database design. In *Proc. 3rd Int. Conf. on Very Large Databases* (Tokyo, Japan, Oct. 6–8, 1977), ACM, New York, 1977, pp. 441–446 Also in *Tutorial: Data Base Management in the 1980s,* J A. Larson and H.A. Freeman, Eds., IEEE, NY, 1981, pp. 269–274.
20. FAGIN, R A normal form for relational databases that is based on domains and keys. *ACM Trans. Database Syst. 6,* 3 (Sept 1981), 387–415.
21. FAGIN, R. Horn clauses and database dependencies. *J. ACM 29,* 4 (Oct. 1982), 952–985. Extended abstract appeared in *Proc 12th Ann. ACM Symp. on Theory of Computing* (Los Angeles, Calif., Apr. 28–30, 1980), ACM, New York, 1980, pp. 123–134.
22 FAGIN, R , MENDELZON, A.O., AND ULLMAN, J.D. A simplified universal relation assumption and its properties. *ACM Trans. Database Syst 7,* 3 (Sept. 1982), 343–360.
23. GOODMAN, N., AND SHMUELI, O Characterizations of tree database schemas. Tech. Rep., Harvard Univ., Cambridge, Mass., 1981.
24 GOODMAN, N., AND SHMUELI, O. Private communication, Jan 1982.
25. GOODMAN, N., AND SHMUELI, O. Tree queries: A simple class of relational queries. *ACM Trans. Database Syst. 7,* 4 (Dec. 1982), 653–677.
26. GRAHAM, M.H. On the universal relation. Tech Rep , Univ. of Toronto, Toronto, Ont., Can., Sept. 1979
27 GRAHAM, M H. Facts about CAG-C database schemas Unpublished manuscript, Sept. 1981.
28 HONEYMAN, P Testing satisfaction of functional dependencies. *J ACM 29,* 3 (July 1982), 668–677.
29 HULL, R Acyclic join dependency and database projections. Tech. Rep., Univ of Southern California, Los Angeles, Calif , June 1981.
30 KAHN, J., KLEITMAN, D., AND LINIAL, W Private communication, Aug. 1982.
31 LIEN, Y.E On the equivalence of database models. *J. ACM 29,* 2 (Apr. 1982), 333–363.
32 MAIER, D., AND ULLMAN, J D. Connections in acyclic hypergraphs. In *Proc. ACM Symp. on*

Principles of Database Systems (Los Angeles, Calif., Mar. 29–31, 1982), ACM, New York, 1982, pp. 34–39.

33. MAIER, D., ULLMAN, J.D , AND VARDI, M Y. The revenge of the JD. In *Proc. 2nd ACM Symp. on Principles of Database Systems* (Atlanta, Ga., Mar 21–23, 1983), ACM, New York, 1983, pp. 279–287.
34. RISSANEN, J. Theory of relations for databases—A tutorial survey. In *Proc. 7th Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 64, J. Winkowski, Ed., Springer-Verlag, New York, pp. 537–551.
35. SAGIV, Y Can we use the universal instance assumption without using nulls? In *Proc. Int. Conf. on Management of Data* (Ann Arbor, Mich., Apr. 29–May 1, 1981), ACM, New York, 1981, pp 108–120.
36. TARJAN, R.E., AND YANNAKAKIS, M. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. Tech. Rep., Bell Labs, Murray Hill, N.J., Mar. 1982.
37. ULLMAN, J.D. The U.R. strikes back. In *Proc. ACM Symp. on Principles of Database Systems* (Los Angeles, Calif., Mar. 29–31, 1982), ACM, New York, 1982, pp. 10–22
38. YANNAKAKIS, M. Algorithms for acyclic database schemes. In *Proc 7th Int. Conf. on Very Large Databases* (Cannes, France, Mar. 29–31, 1982), ACM, New York, 1982, pp. 82–94.
39. YANNAKAKIS, M. Private communication, Sept. 1981.
40. YU, C.T., AND OZSOYOGLU, M.Z. An algorithm for tree-query membership of a distributed query. In *Proc 1979 IEEE COMPSAC*, IEEE, New York, 1979, pp. 306–312.
41. ZANIOLO, C Analysis and design of relational schemata for database systems. Ph D. Dissertation, Univ. of California, Los Angeles, Calif , July 1976, available as Tech. Rep UCLA ENG-7669.

RECEIVED DECEMBER 1981; REVISED JULY 1982, ACCEPTED SEPTEMBER 1982