Integrating Information by Outerjoins and Full Disjunctions[1]

Anand Rajaraman
Jeffrey D. Ullman
Dept. of Computer Science
Stanford University

ABSTRACT

Our motivation is the piecing together tidbits of information found on the "web" into a usable information structure. The problem is related to that of computing the natural outerjoin of many relations in a way that preserves all possible connections among facts. Such a computation has been termed a "full disjunction" by Galindo-Legaria. We are thus led to ask the question of when a full disjunction can be computed by some sequence of natural outerjoins. The answer involves a concept of from Fagin [1983] called "γ-acyclic hypergraphs." We prove that there is a natural outerjoin sequence producing the full disjunction if and only if the set of relation schemes forms a connected, γ-acyclic hypergraph.

I. Motivation

Let us imagine we are constructing an information resource that accepts queries about university information. We gather our information from the on-line information found at the various universities and their schools or departments, and we integrate the information into an object structure. In particular, these structures follow the OEM (Object-Exchange Model) of Papakonstantinou, Garcia-Molina, and Widom [1995] that is used in the Tsimmis information-integration project at Stanford. This model supports tree-structured "objects." Each object has a label and may be atomic or have zero or more subobjects.

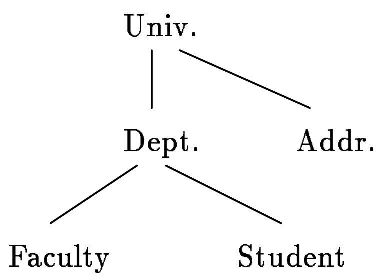

Fig. 1.1. An OEM object structure.

[1] Work supported by USAF contract F33615–93–1–1339, NSF grant IRI–92–23405, and ARO grant DAAH04–95–1–0192.

Example 1.1: Figure 1.1 suggests an object with some subobjects and their labels. That is, it shows that a university object might have one or more address subobjects and some department subobjects. Address objects are atomic, presumably string-valued. Department objects have faculty and student subobjects. □

Elements of information correspond to leaves of such a tree. Thus, in Fig. 1.1, we would use facts such as "Jeff Ullman is a faculty member in the CS Department at Stanford," corresponding to the Faculty leaf in Fig. 1.1, or "the address of Stanford University is Stanford CA 94305," corresponding to the Address leaf. Note that information elements correspond to paths from root to leaf, and there is little utility in partial-path information such as "Jeff Ullman is a faculty member in some CS department" or "Stanford University has a CS department."

Hypergraph Representation of OEM Structures

We may thus think of information elements as corresponding to tuples in relations whose schemes correspond to the labels of root-to-leaf paths in the OEM structure.

Example 1.2: Figure 1.1 corresponds to the database scheme $\{UDF, UDS, UA\}$, where we have used abbreviation U for "University," and appropriate letters for the other labels. □

Database schemes are often profitably represented as hypergraphs, whose nodes are the attributes (equivalently, OEM labels) and whose hyperedges are the relation schemes or sets of attributes; see Fagin, Mendelzon, and Ullman [1982], Ullman [1989], or (for an equivalent notation) Bernstein and Goodman [1981]. For instance, the database scheme implied by Fig. 1.1 is shown in Fig. 1.2.

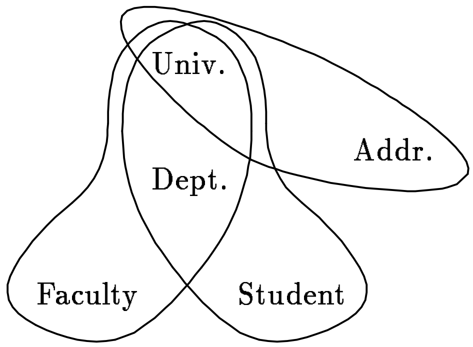

Fig. 1.2. Hypergraph representation of university OEM objects.

Maximizing the Connections Among Facts

Suppose we know that Ullman is in the Stanford CS Dept., and that the address of Stanford is "Stanford CA 94305." We would like to infer that "Stanford CA 94305" is an address for Ullman. In general, if we have any collection of facts that agree on common attributes (are *join-consistent*) we would like them to be available in the "result" of this collection of facts.

This question is very much like computing a universal relation (Maier, Ullman, and Vardi [1984], Maier, Rozenshtein, and Warren [1986], Ullman [1989], e.g). However, the universal-relation work has dealt primarily with exploiting dependencies in the data and with eliminating bogus connections. We believe that at least for the OEM tree-structured objects, any join-consistent connection should be treated as valid. Thus, we want what Galindo-Legaria [1994] calls a *full disjunction*: a relation with nulls (represented by $\perp$) such that every set of join-consistent tuples in our database appears within a tuple of the full disjunction, with either $\perp$ or a concrete value in each attribute not found among our set of tuples.

Example 1.3: Following the notation of Example 1.2, suppose we have tuple udf in relation UDF and tuple ua in relation UA. In the full disjunction, we must find a tuple over the complete set of attributes $UDFSA$ that is either $udf\perp a$ or $udfsa$ for some concrete student name s. □

We note that the relation scheme over which we wish to maximize connections without losing information need not correspond to a hierarchical structure as in the previous examples. In general, the relations may come from either the same or different relational databases as well. For example, it is easy to imagine a relational database with the relation scheme shown in Fig 1.2. A typical application arises in *data warehousing*, where we wish to store a computed view on the base relations that maximizes connections between their tuples.

II. Outerjoins and Full Disjunctions

The *outerjoin* is a variant of the join in which tuples of one relation that do not match any tuple of the second relation are added to the result, padded with nulls. This operation is part of the new SQL92 standard (ANSI [1992]), which has resulted in increased interest in its implementation. Also, as suggested by the examples of Section I, we believe that the outerjoin is more appropriate than the conventional (or "inner") join, because the outerjoin is *monotonic* (the projection of the outerjoin onto its operands' schemes contains the operand relations). However, the join is not monotonic in this sense.

Unfortunately, the outerjoin operator is not associative and therefore does not generalize readily to more than two relations. Thus, when outerjoining three or more relations, we really want the full disjunction, which is commutative, associative, and monotonic. In this section we give formal definitions of the outerjoin and the full disjunction.

Natural Outerjoins

In this paper, we shall consider only the *natural outerjoin*, denoted $\overset{\circ}{\bowtie}$, because it matches the model introduced in Section I. Let R and S be two relations. Since these relations may be the result of outerjoins, it is possible that some tuples in R and/or S contain nulls. Then $R \overset{\circ}{\bowtie} S$ has scheme $R \cup S$ (i.e., the union of the attributes of R and S)[2] and consists of the following tuples:

[2] We use the convention that R and S stand both for the relations themselves — the sets of tuples — and the relation schemes — the sets of attributes. We trust that context will make clear which is meant.

1. All tuples in $R \bowtie S$, the natural join of R and S. To be precise, this join does not allow nulls to be equated. Thus, the tuples t produced in this step are all those over scheme $R \cup S$ such that there are tuples r in R and s in S that agree and are nonnull in all the attributes of $R \cap S$; the resulting tuple t is the one that agrees with r and/or s on each attribute of $R \cup S$.

2. For every tuple r in R that joins with no tuple of S (i.e., $\pi_R(R \bowtie S)$ does not contain r), a tuple with r in the attributes of R and $\perp$ in the attributes of $S - R$.

3. For every tuple s in S that joins with no tuple of R, a tuple with s in the attributes of S and $\perp$ in attributes of $R - S$.

Example 2.1: Continuing with the relations of Examples 1.2 and 1.3, suppose that UDF contains only the tuple *udf*, UA has only the tuple ua, and UDS is empty. Then $UDS \overset{\circ}{\bowtie} UA$ has scheme $UDSA$ and its only tuple is $u\perp\perp a$, formed by rule (2) above. If we then compute $UDF \overset{\circ}{\bowtie} (UDSA)$, we get scheme $UDFSA$ with the two tuples $udf\perp\perp$ and $u\perp\perp\perp a$.

Notice that we do not get a tuple in which *udf* and ua are connected, even though they are join-consistent. The intuitive reason is that by joining UDS and UA first, we extended ua with a $\perp$ in the D attribute prematurely, making it impossible to connect ua with *udf* later. Note that the definition of outerjoin does not permit $\perp$ to match any value, including $\perp$, when taking the natural (inner) join.

However, if we join the three relations in what we shall see later is the "proper" order, $(UDF \overset{\circ}{\bowtie} UDS) \overset{\circ}{\bowtie} UA$, we get tuple $udf\perp$ over scheme $UDFS$ for the first outerjoin and then join this tuple with ua in the second outerjoin to get $udf\perp a$ over scheme $UDFSA$ in the result. □

Tuple Subsumption

We say that tuple t *subsumes* tuple u if t and u agree in every component where u is not $\perp$. That is, t is obtained from u by replacing zero or more nulls by concrete values. For instance, $udf\perp a$ subsumes $u\perp\perp\perp a$.

Connected Hypergraphs

Normally, we shall be interested in collections of relations whose schemes form a connected hypergraph. While the intuitive meaning of "connected" should be obvious, let us formally define a hypergraph to be *disconnected* if we can partition its hyperedges into two nonempty sets such that no node appears in members of both sets. Otherwise, the hypergraph is *connected.*

Full Disjunctions

Let $\mathcal{R} = R_1, R_2, \ldots, R_n$ be relations whose tuples do not have nulls. We say R is the *full disjunction* for $\mathcal{R}$ if the following hold:

A. *No redundancy*: No tuple of R subsumes any other tuple of R.

B. *Tuples of R come from connected pieces of $\mathcal{R}$*: Let t be a tuple of R. Then there is some connected subset of the relations of $\mathcal{R}$ such that t, restricted to its nonnull components, is the join of tuples from those relations.

C. *All connections are represented*:

1. Let $t_1, \ldots, t_k$ be tuples chosen from distinct relations $R_{i_1}, \ldots, R_{i_k}$, respectively, such that $\{R_{i_1}, \ldots, R_{i_k}\}$ is a connected hypergraph.
2. Let the t_i's be join-consistent, in the sense that for any attribute A, all the components among the t_i's corresponding to attribute A have the same value.
3. Let t be the tuple that agrees with each of the t_i's in those attributes appearing among any of $R_{i_1}, \ldots, R_{i_k}$ and that has $\perp$ in other attributes found among the schemes of $\mathcal{R}$.

Then t is subsumed by some tuple of R.

Example 2.2: For the relations of Example 2.1, the singleton relation $R = \{udf\perp a\}$ is the full disjunction. Clearly (A) is satisfied since there is only one tuple. (B) is satisfied since the nonnull components of the tuple $udf\perp a$ is the join of the tuples udf from UDF and ua from UA. Finally, condition (C) is also satisfied. The only nontrivial combination of tuples is udf and ua. The tuple t constructed as in (C3) is $udf\perp a$, which is subsumed by a tuple in R, in fact by the identical tuple. □

The following theorem is useful to know, although it is not used directly in the results to follow.

Theorem 2.1: The full disjunction is unique.

Proof: It is easy to see that the only relation R satisfying the definition above consists of all those tuples t such that:

a) t is the join of some collection of join-consistent tuples from a connected subset of the relations $R_1, \ldots, R_n$, padded out with nulls, and

b) There is no additional relation among the R_i's with a tuple that is join-consistent with t. □

The set of tuples described in the proof of Theorem 2.1 is exactly what Lien [1982] calls *maxtrav* (maximal traversals). He shows the equivalence of maxtrav to the set of tuples we described by conditions (A–C) above. Lein [1982] also gives a result (equivalence to "network database schemes") about the same class of hypergraphs that we consider in this paper — connected, γ-acyclic hypergraphs.

Following Theorem 2.1, we shall refer to "the full disjunction." We shall use the notation $\text{FD}(\mathcal{R})$ for the full disjunction of some hypergraph or collection of relations $\mathcal{R}$.

Computing the Full Disjunction

We would like to find some simple way of computing the full disjunction of a set of relations. The problem was studied by Galindo-Legaria [1994], but only for the case of non-natural (or "theta") outerjoins, where all attributes of both relations are kept in the result, and connections are represented by predicates on pairs of attributes, one from each relation.

In general, the full disjunction, as its name implies, can be computed by

1. Taking the union of all joins of connected subsets of the relations,
2. Padding with nulls, and
3. Deleting subsumed tuples.

However, that is clearly too expensive to be desirable. Thus, Galindo-Legaria [1994] gave a test for when some order of outerjoins is guaranteed to produce the full disjunction by itself. This test is simple. Create a graph whose nodes are the relations and whose edges connect relations that are constrained by one or more comparison; if the graph is acyclic then an outerjoin order yielding the full disjunction exists, and if not, then not.[3]

However, this test does not apply to the natural outerjoin. In our running example of relations UDF, UDS, and UA, there are equality constraints on U between each pair of relations, giving a graph that is a triangle and therefore cyclic in the sense of Galindo-Legaria. Yet as we shall see, these three relations do have an outerjoin order that yields the full disjunction: $(UDF \overset{\circ}{\bowtie} UDS) \overset{\circ}{\bowtie} UA$, although, as was suggested by Example 2.1, the other two orders, $(UDF \overset{\circ}{\bowtie} UA) \overset{\circ}{\bowtie} UDS)$ and $(UDS \overset{\circ}{\bowtie} UA) \overset{\circ}{\bowtie} UDF$ do not in general yield the full disjunction.[4]

Thus, we define a *sound outerjoin ordering* for a database scheme $\mathcal{R} = \{R_1, \ldots, R_n\}$ to be an expression in which

1. Each relation appears exactly once,
2. The only operator is $\overset{\circ}{\bowtie}$, and
3. The result is $\text{FD}(\mathcal{R})$.

The balance of this paper characterizes those database schemes that have a sound outerjoin ordering.

III. Gamma-Acyclic Hypergraphs

The class of γ-acyclic hypergraphs is one of several kinds of "acyclic" hypergraphs studied by Fagin [1983]. It is a special case of the more common class of "acyclic hypergraph" of Graham [1979], Yu and Ozsoyoglu [1979], Bernstein and Goodman [1981], or Fagin, Mendelzon, and Ullman [1982], which Fagin [1983] calls α-acyclic,

Fagin [1983] gives several equivalent definitions of γ-acyclicity. We shall only give here those definitions that are of use in our characterization of the hypergraphs for which a full disjunction can be computed by outerjoins in some order; those are exactly the connected γ-acyclic hypergraphs.

---

[3] Do not confuse this graph and the notion of "acyclicity" with the hypergraphs and the concept of hypergraph acyclicity suggested by Fig. 1.2.

[4] Note that the outerjoin is commutative although not associative, so these are the only three different ways the outerjoin of three relations can be taken.

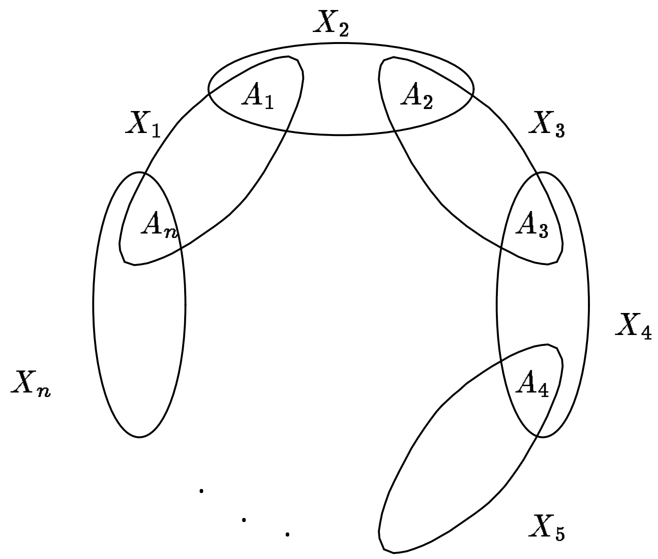

Fig. 3.1. A pure cycle.

Pure Cycles

A *pure cycle* is a collection of $n \geq 3$ hyperedges and nodes suggested by Fig. 3.1. Each pair of hyperedges X_i and X_{i+1} (and also X_n and X_1) have at least one node A_i (or A_n in the case of X_n and X_1) in common. Moreover, none of the shared nodes A_j appears in more than the two hyperedges of the pure cycle suggested by Fig. 3.1. These nodes may appear in hyperedges that are ***not*** part of the pure cycle, however. There may be more than one node in the intersections shown, but those nodes must not appear in any other hyperedges of the pure cycle.

Gamma-Three-Cycles

A *γ-3-cycle* is a set of three hyperedges in the configuration of Fig. 3.2. That is, the nodes A, B, and C must exist; any other regions of the diagram may or may not be empty, and the regions containing A, B, and C may also contain other nodes. Put another way, there must be some node in all three hyperedges, one that is in only X and Z, and one that is only in Y and Z. For example, the smallest γ-3-cycle consists of the three hyperedges $\{ABC, AB, BC\}$.

Gamma-Cycles

A *γ-cycle* is a cycle of at least three edges, in the form of Fig. 3.1. However, unlike the pure cycle, a γ-cycle permits A_n to appear in any of hyperedges $X_2, \ldots, X_{n-1}$, as well as X_1 and X_n. All other nodes $A_1, \ldots, A_{n-1}$ in the intersections appear only in those X_i's shown.

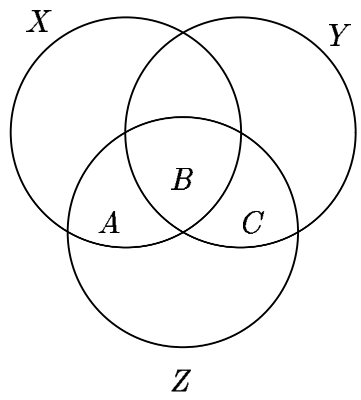

Fig. 3.2. A γ-3-cycle.

Definition of Gamma-Acyclic Hypergraphs

Fagin [1983] defines a hypergraph to be *γ-acyclic* if and only if it has no γ-cycle. Equivalently, a hypergraph is γ-acyclic if and only if it contains no pure cycle and no γ-3-cycle. A hypergraph is *γ-cyclic* if and only if it is not γ-acyclic.

Example 3.1: The hypergraph of Fig. 1.2 is γ-acyclic. For one argument, note that there could not be a γ-3-cycle. If there were, only U could be the node in all three edges. Then, D could serve as one of the nodes in exactly two of these edges. However, none of the other three nodes is in two of the three hyperedges. Also, it is easy to see that there is no pure cycle in this hypergraph. □

Fagin [1983] also proves a characterization of γ-acyclic hypergraphs by *Bachman diagrams*. Given a hypergraph $\mathcal{H}$, we may add to it as additional hyperedges all intersections of two or more hyperedges of $\mathcal{H}$. Let the resulting hypergraph be $\mathcal{G}$. We then create an (ordinary) graph whose nodes are the hyperedges of $\mathcal{G}$ and for which there is an arc from node E to node F if $E \subset F$ and this containment is as close as possible; that is, for no edge G is $E \subset G \subset F$. The resulting graph is the Bachman diagram of $\mathcal{H}$, denoted $\mathrm{BD}(\mathcal{H})$.

A Bachman diagram is *acyclic* if, treated as an undirected graph, there are no cycles. Acyclic Bachman diagrams were studied by Lein [1982] and Yannakakis [1982]. Fagin [1983] proves that hypergraph $\mathcal{H}$ is γ-acyclic if and only if $\mathrm{BD}(\mathcal{H})$ is acyclic.

Example 3.2: The Bachman diagram of the hypergraph in Fig. 1.2 is shown in Fig. 3.3. Note that for a tree scheme like Fig. 1.2, the Bachman diagram looks almost like the original tree of Fig. 1.1, if we draw the smaller sets above the larger. In fact, they will be isomorphic unless there are nonbranching paths in the tree scheme, i.e., an attribute that has only one child. Then, the nonbranching paths will be collapsed to nodes in the Bachman diagram. □

Example 3.3: Consider the γ-acyclic hypergraph with hyperedges $\{AB, BC, CD\}$. Its Bachman diagram is shown in Fig. 3.4. Note that this graph is a tree in the undirected sense, although it has two "roots" in the directed sense. □

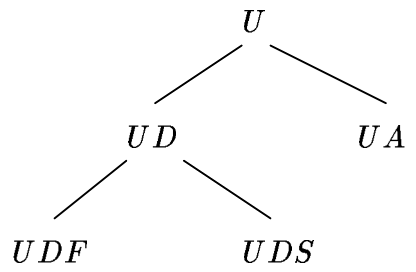

Fig. 3.3. Bachman diagram for Fig. 1.2.

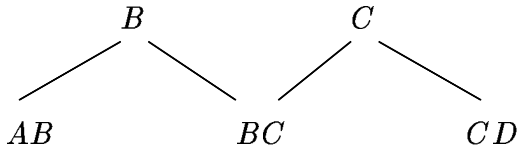

Fig. 3.4. Bachman diagram for $\{AB, BC, CD\}$.

IV. Gamma Acyclicity is Necessary for Sound Outerjoin Orderings

The main result of this paper is that γ-acyclicity characterizes the hypergraphs for which there is a sound outerjoin ordering. Of course every hypergraph can have the full disjunction computed by *some* expression. We are interested here in only the simplest expressions, where just the natural outerjoin operator is used. In this section we prove that the γ-cyclic hypergraphs do not have a sound outerjoin ordering. In the next section we show how to compute the full disjunction with a sound outerjoin ordering for any connected γ-acyclic hypergraph.

Our plan for the necessity proof is to consider separately those hypergraphs that have a γ-3-cycle and those that have a pure cycle. For each case, we show how to construct a counterexample. That is, we show for each possible outerjoin ordering how to find a particular database for which the ordering is not a sound outerjoin ordering (i.e., it does not produce the full disjunction).

Gamma-Three-Cycles Do Not Have Sound Outerjoin Orderings

We begin with a technical lemma that gives us needed counterexamples.

Lemma 4.1: Suppose relations R, S, and T are such that

1. There is an attribute A in the schemes of relations S and T that is not in the scheme for relation R, and
2. There is an attribute B that is in both R and T (and possibly S).

Then $(R \stackrel{\circ}{\bowtie} S) \stackrel{\circ}{\bowtie} T$ is not a sound outerjoin ordering.

Proof: Consider the database where R and T each contain a single tuple r and t, respectively, and these tuples are join-consistent; that is, they agree in B and any other common attributes. Let relation S be $\emptyset$. Then $R \stackrel{\circ}{\bowtie} S$ is the tuple r padded with $\perp$ in each attribute of $S - R$. In particular, this tuple has $\perp$ in attribute A. Thus, the tuple of $R \stackrel{\circ}{\bowtie} S$ does not join with t, because they disagree on A. As a result, $(R \stackrel{\circ}{\bowtie} S) \stackrel{\circ}{\bowtie} T$ contains two tuples: t padded with $\perp$'s in attributes other than those of T, and r, also padded with $\perp$'s. However, the full disjunction is the single tuple that agrees with r and t on attributes of either R or T and has $\perp$ in any other attributes. Thus, the expression $(R \stackrel{\circ}{\bowtie} S) \stackrel{\circ}{\bowtie} T$ does not produce the full disjunction in this case and is not a sound outerjoin ordering. □

Example 4.1: Let us see two examples of how Lemma 4.1 can be used in practice. Consider the relations AB, BC, and ABC. These form the minimal γ-3-cycle, as in Fig. 3.2. By symmetry, there are two significantly different orders in which we could take the outerjoin:

1. $(AB \stackrel{\circ}{\bowtie} ABC) \stackrel{\circ}{\bowtie} BC$
2. $(AB \stackrel{\circ}{\bowtie} BC) \stackrel{\circ}{\bowtie} ABC$

All other orders can be obtained either by commutativity of outerjoin and/or by interchanging AB with BC.

For expression (1), Lemma 4.1 applies because there is an attribute C that is in ABC and BC, but not in AB. Also, AB and BC have an attribute, B, in common. Thus, we expect that expression (1) is not a sound outerjoin ordering. Specifically, consider the database $AB = \{ab\}$, $BC = \{bc\}$, and $ABC = \emptyset$. The full disjunction is $\{abc\}$, but expression (2) produces $\{ab\perp, \perp bc\}$. Thus indeed we see that order (1) is not a sound outerjoin ordering.

Similarly, order (2) is not a sound outerjoin ordering according to Lemma 4.1. In confirmation, consider the following relations: $AB = \{ab\}$, $BC = \emptyset$, and $ABC = \{abc\}$. Evidently, the full disjunction is $\{abc\}$. However, when we join AB with BC we get the tuple $ab\perp$. Thus, when we join this result with ABC, there is no match between tuples $ab\perp$ and abc, because c cannot match $\perp$. Thus, the result is the two tuples $\{ab\perp, abc\}$. Since one tuple is subsumed by the other, we have not computed the full disjunction. □

The second ordering in Example 4.1 presents the weakest argument, since except for the "no redundancy" condition (A) in the definition of full disjunction, the result would be satisfactory. That is, abc "looks like" it came from both ab and abc; it just happens that the second outerjoin in expression (2) is incapable of joining those two tuples together. Our argument for imposing condition (A) is not just to make our characterization exact. Eliminating subsumed tuples is an expensive operation, even more than eliminating duplicates in conventional relations. Thus, it is reasonable to require "good" methods for computing the full disjunction to avoid the need for elimination steps.

Lemma 4.2: No γ-3-cycle has a sound outerjoin ordering.

Proof: Consider the diagram of Fig. 3.2. If there were a sound outerjoin ordering for the three relations X, Y, and Z, one of them would have to come last. If Z is last, then attribute B of Fig. 3.2 can play the role of B in Lemma 4.1, and A or C of Fig. 3.2 can play the role of A in Lemma 4.1, telling us we do not have a sound outerjoin ordering.

If X is last, then we can write the expression as $(Y \overset{\circ}{\bowtie} Z) \overset{\circ}{\bowtie} X$. Note that the outerjoin is commutative, so this expression is equivalent to $(Z \overset{\circ}{\bowtie} Y) \overset{\circ}{\bowtie} X$; i.e., we may choose whichever of Y and Z we wish to put in the role of S of Lemma 4.1. Given our choice, the attribute A in expression $(Y \overset{\circ}{\bowtie} Z) \overset{\circ}{\bowtie} X$ plays the role of A in Lemma 4.1 and B plays the role of B, again showing we do not have a sound outerjoin ordering.

Finally, Y could come last in the outerjoin expression. This case is symmetric to the case where X comes last. $\square$

Now we must extend Lemma 4.2 to all hypergraphs that include a γ-3-cycle among their hyperedges. First, we need a technical lemma that says, in effect, that if we throw an empty relation into an outerjoin expression, we cannot turn an unsound outerjoin ordering into a sound one.

Lemma 4.3: Suppose R and S are two relations, and let R' be R with its tuples extended with nulls in one or more attributes that do not appear in the scheme of R (i.e., R' could be the result of outerjoining R with some empty relation). Then:

a) Every tuple in $R' \overset{\circ}{\bowtie} S$ is subsumed by some tuple in $R \overset{\circ}{\bowtie} S$, extended with nulls to attributes of $R' - R - S$.

b) If t_1 and t_2 are tuples of $R \overset{\circ}{\bowtie} S$, and t_1 *properly subsumes* t_2 (that is, t_1 subsumes t_2 and is not equal to t_2), then $R' \overset{\circ}{\bowtie} S$ has a pair of tuples one of which properly subsumes the other. We shall call such a pair a *subsuming pair.*

Proof:

PART (a): Suppose that t is in $R' \overset{\circ}{\bowtie} S$. There are three cases, depending on whether

1. t is the join of r' from R' and s from S,
2. t is a tuple of R' padded with nulls, or
3. t is a tuple of S padded with nulls.

Case 1: Let r be r' with nulls in the attributes of $R' - R$ removed. Then r is join-compatible with s. Thus, t is some tuple u in $R \overset{\circ}{\bowtie} S$, padded with nulls in attributes of $R' - R - S$.

Case 2: Here, t must be $\perp$ in every attribute outside of R. Let r be t restricted to attributes of R. Then either r, padded with nulls, is in $R \overset{\circ}{\bowtie} S$, or r joins with some s in S. In the first subcase, t is clearly in the relation formed by padding $R \overset{\circ}{\bowtie} S$ with nulls in attributes of $R' - R - S$. In the second subcase, r joined with s and padded subsumes t.

Case 3: If t is a padded tuple from S, the argument that t is subsumed is similar to that of Case (2).

PART (b): In what follows, let us use $x > y$ for "x properly subsumes y." There are three cases, depending on how t_1 got into $R \overset{\circ}{\bowtie} S$.

1. t_1 is the join of tuples r_1 from R and s_1 from S.
2. t_1 is a tuple of R padded with nulls.
3. t_1 is a tuple of S padded with nulls.

Case 1: There are two subcases, depending on whether $R' - R$ contains any attributes of S. If not, it is easy to see that the padding of tuples from R to make tuples of R' has no effect on what tuples of R or R' join with tuples of S. Thus, t_1 and t_2, padded, will be in $R' \stackrel{\circ}{\bowtie} S$, proving the lemma for this subcase.

If $R' - R$ includes an attribute of S, then no tuple of R' can join with any tuple of S. We thus know that r_1 and s_1, padded, will be in $R' \stackrel{\circ}{\bowtie} S$. If t_2 comes from R, then r_1 (padded) $> t_2$, and t_2 padded will be in $R' \stackrel{\circ}{\bowtie} S$. Thus, $R' \stackrel{\circ}{\bowtie} S$ contains a subsuming pair (r_1 and t_2, padded). Similarly, if t_2 comes from S the analogous argument holds. Finally, if t_2 is the join of r_2 from R and s_2 from S, then either $r_1 > r_2$ or $s_1 > s_2$ or both. Both r_2 and s_2, padded, will be in $R' \stackrel{\circ}{\bowtie} S$, and we again conclude that $R' \stackrel{\circ}{\bowtie} S$ contains a subsuming pair.

Case 2: t_1 comes from R. If t_2 also comes from R, it is easy to show that R' has a subsuming pair, and therefore so does $R' \stackrel{\circ}{\bowtie} S$. If t_2 were the join of tuples from R and S, then t_1 would also join with this tuple from S, because the joining tuples would have to agree and be nonnull in attributes of $R \cap S$. Thus, we can rule out the possibility that t_2 comes from both R and S.

Finally, suppose t_2 comes from S. We know that t_1 (padded) is in $R' \stackrel{\circ}{\bowtie} S$. Since t_2 does not join with any tuple of R, it surely does not join with a tuple of R', and therefore t_2 (padded) is in $R' \stackrel{\circ}{\bowtie} S$. Thus, t_1 and t_2 (padded) form a subsuming pair in $R' \stackrel{\circ}{\bowtie} S$.

Case 3: similar to case (2). We argue that if t_1 comes from S, then t_2 cannot be the join of tuples from R and S. Whether t_2 comes from R or S, it will appear, padded, in $R' \stackrel{\circ}{\bowtie} S$ and therefore $t_1 > t_2$ will be our subsuming pair. $\square$

Lemma 4.4: No hypergraph with a γ-3-cycle has a sound outerjoin ordering.

Proof: Suppose not; i.e., hypergraph $\mathcal{H}$ has both a γ-3-cycle and a sound outerjoin ordering, E. Let us assign relations to the operands of E (hyperedges of $\mathcal{H}$) so that the three operands corresponding to the γ-3-cycle are given appropriate relations according to Lemma 4.2. The exact relations chosen depends on the order in which the three operands are grouped in E, but whatever the order there is some suitable assignment of two singleton relations and one empty relation. Assign all other operands $\emptyset$ for their relations.

Let F be E with all operands other than the three of the γ-3-cycle deleted. We know by Lemma 4.2 that the result R of F is not the full disjunction; either a tuple of the full disjunction is subsumed by no tuple of R or (as in Case 2 of Example 4.1) R includes two tuples, one of which properly subsumes the other.

In the first case, where a connection that should be in the full disjunction is missing in R, Lemma 4.3(a) tells us that adding the empty relations at various points in E cannot produce a tuple that is not subsumed by a tuple of R, suitably padded with nulls. Thus, E also fails to produce the full disjunction for its arguments.

In the second case, where a subsuming pair exists in R, Lemma 4.3(b) assures us that a subsuming pair also exists when E is evaluated on the arguments defined above.

Thus again E fails to produce the full disjunction. We conclude that E was not a sound outerjoin ordering as assumed. □

Pure Cycles Do Not Have Sound Outerjoin Orders

Now, let us consider a hypergraph that is a pure cycle as suggested by Fig. 3.1.

Lemma 4.5: No hypergraph that is a pure cycle has a sound outerjoin ordering.

Proof: In what follows, we shall interpret subscript i in X_i in the "end-around" sense. That is, X_i refers to that X_j such that $1 \leq j \leq n$ and i and j leave the same remainder when divided by n. Thus, X_0 is synonymous with X_n, X_{n+1} means X_1, and so on. Subscripts for the attributes A_i will be treated the same way.

Suppose this hypergraph has a sound outerjoin ordering E. In E, the final outerjoin has two arguments, one of which is the outerjoin of some set $\mathcal{S}_1$ of the hyperedges $X_1, X_2, \ldots, X_n$ and the other is the outerjoin of the other hyperedges, say $\mathcal{S}_2$. Without loss of generality, assume that $\mathcal{S}_2$ has at least two members; the fact that $n \geq 3$ assures us that at least one set can play the role of $\mathcal{S}_2$.

Thus, we can find some sequence of one or more hyperedges in $\mathcal{S}_1$, say $X_i, \ldots, X_j$, such that neither X_{i-1} nor X_{j+1} are in $\mathcal{S}_1$. Now, consider a database where $X_i, \ldots, X_{j+1}$ each contain one tuple, and these tuples are join-compatible. All other relations are $\emptyset$. Then the full disjunction of these relations contains the one tuple that is the join of all the tuples in $X_i, \ldots, X_{j+1}$, padded with nulls.

Now consider what happens when we evaluate E with these relations. The result of the outerjoin of the relations in $\mathcal{S}_2$ cannot contain a tuple that is nonnull in attribute A_{i-1}, an attribute in the intersection of X_{i-1} and X_i. The reason is that since $\mathcal{S}_2$ has at least two members, i cannot be $j-2$ (modulo n). Thus, X_{i-1} is not X_j or X_{j+1}, and hence A_{i-1} is not in X_{j+1}.

As a consequence, when we apply the last outerjoin of E, we cannot get a single tuple. Whatever tuple or tuples the $\mathcal{S}_1$ side has produced from $X_i, \ldots, X_j$, at least one of these tuples has a nonnull value in A_{i-1} (ideally, only one tuple, the join of all the tuples from $X_i, \ldots, X_j$ is produced, but there could be more if the $\mathcal{S}_1$ side of E itself fails to produce the full disjunction of its relations). Since this tuple cannot join with some tuple from $\mathcal{S}_2$, the result of E cannot contain only one tuple. Therefore, E does not produce the full disjunction for this database. □

Lemma 4.6: No hypergraph that contains a pure cycle has a sound outerjoin ordering.

Proof: The proof is analogous to Lemma 4.4. If there is such a hypergraph with a sound outerjoin ordering E, construct from E another expression F that is E with all operands not in the pure cycle deleted. Construct a database with a counterexample in the pure cycle and $\emptyset$ for all other relations.

By Lemma 4.5, we know that F does not produce the full disjunction for its operands. By Lemma 4.3, the inclusion of the empty relations cannot change F into an expression that computes the full disjunction for its operands. Thus, E fails to produce its full disjunction and is not a sound outerjoin ordering. □

Completing the Necessity Proof

By Fagin [1983], every γ-acyclic hypergraph contains as a subhypergraph either a γ-3-cycle or a pure cycle. Lemmas 4.4 and 4.6 show that hypergraphs containing either do not have sound outerjoin orderings. Thus we have proved:

Theorem 4.1: If a hypergraph has a sound outerjoin ordering, then it must be γ-acyclic. $\square$

V. Gamma Acyclicity is Sufficient for Sound Outerjoin Orderings

Now, let us prove the converse: for every γ-acyclic hypergraph there is a sound outerjoin ordering. The heart of the proof is the notion of "γ-decomposition": a way to split a hypergraph into two pieces. If the original hypergraph is γ-acyclic, then the pieces are also γ-acyclic (the existence of a γ-cycle in one of the pieces implies a γ-cycle in the whole). We recursively produce the full disjunction of the pieces and use the definition of γ-decomposition (which appears below) to claim that the full disjunction of the original hypergraph is the outerjoin of the full disjunctions of the pieces.

A Problem With Disconnected Hypergraphs

Before proceeding, we must note an additional condition: no disconnected hypergraph has a sound outerjoin ordering. The reason is that if the hypergraph is disconnected, then at some point in an outerjoin expression we must take a Cartesian product. The product connects all tuples of one component with all tuples of another, thus giving us too many connections.

When we need to combine two or more relations with disjoint sets of attributes, the appropriate operation is the *outerunion*, that is, the tuples of each relation padded with nulls in the attributes of the other relation(s). With that addition to our repertoire, we can handle all γ-acyclic hypergraphs. We construct a sound outerjoin ordering for each connected component and, if we wish to combine them, use the outerunion operator.

Gamma-Decompositions

Let $\mathcal{H}$ be a connected hypergraph. Then $(\mathcal{H}_1, \mathcal{H}_2)$ is a *γ-decomposition* of $\mathcal{H}$ if:

1. $\mathcal{H}_1$ and $\mathcal{H}_2$ are nonempty, disjoint sets of hyperedges of $\mathcal{H}$ that together include all the hyperedges of $\mathcal{H}$ (i.e., $\mathcal{H}_1$ and $\mathcal{H}_2$ partition the hyperedges of $\mathcal{H}$).

2. The nodes of $\mathcal{H}_1$ and $\mathcal{H}_2$ are each the union of the hyperedges of these hypergraphs (i.e., $\mathcal{H}_1$ and $\mathcal{H}_2$ are hyperedge-generated).

3. Let X be the set of nodes that are in both $\mathcal{H}_1$ and $\mathcal{H}_2$. Then

   a) X is not empty, and

   b) Every hyperedge of $\mathcal{H}$ either contains X or is disjoint from X.

Example 5.1: Consider the hypergraph $\mathcal{H}$ of Fig. 1.2. The decomposition in which $\mathcal{H}_1 = \{UDF, UDS\}$ and $\mathcal{H}_2 = \{UA\}$ is a γ-decomposition. In proof, the intersection of the node sets is U. Every hyperedge of $\mathcal{H}$ either "contains U or is disjoint from it"; in particular each of the hyperedges contains U.

However, $\mathcal{H}_1 = \{UDF\}$ and $\mathcal{H}_2 = \{UDS, UA\}$ is not a γ-decomposition. To see why, note that the intersection of the sets of nodes is UD. But hyperedge UA neither contains UD nor is disjoint from it. $\Box$

The importance of γ-decompositions is in the next lemma, which says that we can get the full disjunction for a hypergraph by decomposing it, recursively computing the full disjunctions of the pieces, and combining them with the outerjoin operator.

Lemma 5.1: Let $(\mathcal{H}_1, \mathcal{H}_2)$ be a γ-decomposition of a connected hypergraph $\mathcal{H}$, and suppose $R_1 = \text{FD}(\mathcal{H}_1)$ and $R_2 = \text{FD}(\mathcal{H}_2)$. Then $R_1 \overset{\circ}{\bowtie} R_2 = \text{FD}(\mathcal{H})$.

Proof: Let X be the intersection of the nodes of $\mathcal{H}_1$ and $\mathcal{H}_2$. We show that tuple t is in $R_1 \overset{\circ}{\bowtie} R_2$ if and only if it is in $\text{FD}(\mathcal{H})$.

IF: First, consider a tuple t in $\text{FD}(\mathcal{H})$. Let $\mathcal{G}$ be the connected subset of the hyperedges of $\mathcal{H}$ such that t is formed by the join of one tuple from the relation corresponding to each hyperedge in $\mathcal{G}$, padded with nulls if necessary. The situation is suggested in Fig. 5.1.

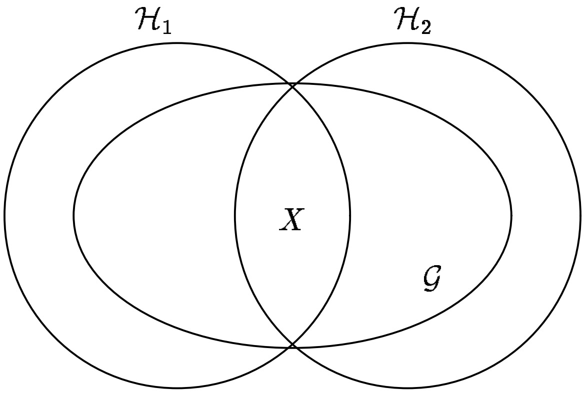

Fig. 5.1. Diagram for Lemma 5.1.

Let t_i be the restriction of t to the nodes of $\mathcal{H}_i$, for $i = 1, 2$. Then t_i is the join of the relations corresponding to the hyperedges in $\mathcal{G} \cap \mathcal{H}_i$, for $i = 1, 2$. Since $\mathcal{G}$ is connected, and the only way hyperedges in $\mathcal{H}_1$ and $\mathcal{H}_2$ connect is through the set of nodes X, it follows that $\mathcal{G} \cap \mathcal{H}_i$ must be connected for $i = 1, 2$ (the possibility that one of these hypergraphs is empty is not ruled out).

By hypothesis, t_i, padded with nulls if necessary, is in relation R_i, for $i = 1, 2$. Let us first suppose t_1 is all nulls; that is, $\mathcal{G} \subseteq \mathcal{H}_2$. Then $t = t_2$ (padded), and when we take the outerjoin $R_1 \overset{\circ}{\bowtie} R_2$, t_2 is padded with nulls to create the tuple t. Similarly, if t_2 is all nulls, then $t = t_1$ (padded), and t is again seen to be in $R_1 \overset{\circ}{\bowtie} R_2$.

Now, suppose neither t_1 nor t_2 is all nulls; i.e., $\mathcal{G}$ intersects both $\mathcal{H}_1$ and $\mathcal{H}_2$. Since $\mathcal{G}$ is connected, there must be some hyperedge E_1 of $\mathcal{G} \cap \mathcal{H}_1$ and some other hyperedge E_2 in $\mathcal{G} \cap \mathcal{H}_2$ that each contain the set of attributes X. Thus, t_1 and t_2 each are nonnull in the attributes of X. Of course, since they each come from t, they agree on these attributes, so in $R_1 \overset{\circ}{\bowtie} R_2$ they join to make t. We have thus proved that $\text{FD}(\mathcal{H}) \subseteq R_1 \overset{\circ}{\bowtie} R_2$.

ONLY IF: The converse is easy. Suppose tuples t_1 and t_2 — from R_1 and R_2, respectively — join. Then t_i is in $\text{FD}(\mathcal{H}_i)$, for $i = 1, 2$. That is, t_i is the join of some maximal set of tuples chosen from the relations corresponding to connected sets of hyperedges of $\mathcal{H}_i$, for $i = 1, 2$. Since these sets of hyperedges must each include one that contains X (or else t_1 and t_2 could not join in $R_1 \overset{\circ}{\bowtie} R_2$), the union of these sets of hyperedges is also connected.

Let t be the tuple that agrees with t_1 and t_2 in all attributes. Then t must be maximal, in the sense that we cannot extend its nonnull components by joining in some tuple of another relation. For if we could do so, then either t_1 or t_2 would not have been maximal, and thus not in their respective full disjunctions. Thus, t is in $\text{FD}(\mathcal{H})$, proving $R_1 \overset{\circ}{\bowtie} R_2 \subseteq \text{FD}(\mathcal{H})$. $\square$

Gamma Reductions

Now we show how to γ-decompose every γ-acyclic hypergraph into single hyperedges. The recursive process of performing γ-decompositions until no more γ-decompositions are possible is called *γ-reduction.* We shall show that for γ-acyclic hypergraphs, γ-reduction leads to a collection of singleton hypergraphs (hypergraphs with one hyperedge). We then use the γ-reduction to produce a sound outerjoin ordering for every connected γ-acyclic hypergraph. Our first job is to show that we can take a single step of γ-decomposition whenever we have to.

Lemma 5.2: Every connected γ-acyclic hypergraph of more than one hyperedge has a γ-decomposition.

Proof: Consider connected γ-acyclic hypergraph $\mathcal{H}$. Construct its Bachman diagram, which must be acyclic. Choose X to be some minimal node of the Bachman diagram, that is, a set of nodes of $\mathcal{H}$ such that no proper subset is a node of the Bachman diagram.

We claim that every hyperedge of $\mathcal{H}$ either contains X or is disjoint from it. For if E were a hyperedge for which $E \cap X$ were neither $\emptyset$ nor X, then $E \cap X$ would be a node of the Bachman diagram and therefore X would not be minimal. Thus X can play the role required of it in condition (3) of the definition of "γ-decomposition." There are two cases, depending on whether or not the removal of X from the Bachman diagram partitions that graph into more than one connected component.

CASE 1: X has only one adjacent node in the Bachman diagram; i.e., the Bachman diagram looks like Fig. 5.2. Now X must be a hyperedge of $\mathcal{H}$ (rather than the intersection of hyperedges), or else there would be no reason to have X in the Bachman diagram. Thus, we may choose $\mathcal{H}_1 = \{X\}$ and $\mathcal{H}_2$ equal to all the other hyperedges. Some hyperedge of $\mathcal{H}_2$ must contain X, or else $\mathcal{H}$ is not connected. Thus, $(\mathcal{H}_1, \mathcal{H}_2)$ meets all the conditions of a γ-decomposition.

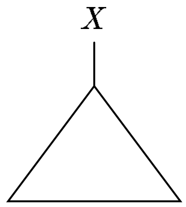

Fig. 5.2. A minimal node X with one child.

CASE 2: Removal of X divides the Bachman diagram into two or more connected components, as suggested in Fig. 5.3. Pick any one of the components, and let $\mathcal{H}_1$ be the set of hyperedges of $\mathcal{H}$ among this component (recall that some nodes of the Bachman diagram may represent intersections of hyperedges, rather than the hyperedges themselves). Pick X and the hyperedges among the remaining components as $\mathcal{H}_2$. Again, the γ-decomposition decomposition conditions are clearly met. □

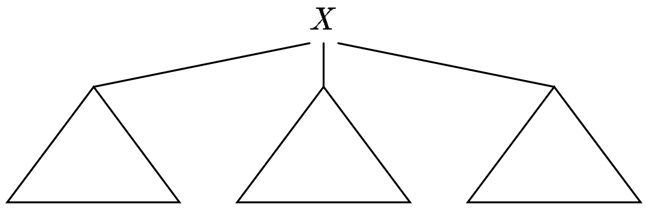

Fig. 5.3. A minimal node X with several children.

Constructing a Sound Outerjoin Ordering

We can now use Lemmas 5.1 and 5.2 to construct the sound outerjoin ordering for a connected γ-acyclic hypergraph.

Algorithm SOJO:

INPUT: A connected, γ-acyclic hypergraph $\mathcal{H}$.

OUTPUT: A sound outerjoin ordering for $\mathcal{H}$.

METHOD: Begin by constructing the Bachman diagram for $\mathcal{H}$.[5] Recursively partition $\mathcal{H}$ and BD($\mathcal{H}$) by selecting minimal nodes of the Bachman diagram as suggested in Lemma 5.2.

BASIS: For the basis, one hyperedge, return the relation for that hyperedge; it is clearly FD($\mathcal{H}$).

INDUCTION: For the induction, suppose $\mathcal{H}$ has $m > 1$ hyperedges.

[5] M. Yannakakis (private communication, July, 1995) points out that the Bachman diagram construction for a γ-acyclic hypergraph takes only linear time.

1. Find a γ-decomposition of $\mathcal{H}$, say $(\mathcal{H}_1, \mathcal{H}_2)$. Such a decomposition can be found by the method of Lemma 5.2.

2. Partition $\text{BD}(\mathcal{H})$ into two pieces corresponding to $\mathcal{H}_1$ and $\mathcal{H}_2$. In Case 1 of Lemma 5.2, $\text{BD}(\mathcal{H}_1)$ is X alone, and $\text{BD}(\mathcal{H}_2)$ is all the other nodes. In Case 2 of Lemma 5.2 we simply remove X, separate the components of the Bachman diagram, and include X in $\text{BD}(\mathcal{H}_2)$ if needed. That is, if X is a hyperedge of $\mathcal{H}$, it goes in $\mathcal{H}_2$. If $\mathcal{H}_2$ is formed from more than one connected component, then again X goes in $\text{BD}(\mathcal{H}_2)$. Otherwise, X is in neither $\text{BD}(\mathcal{H}_1)$ nor $\text{BD}(\mathcal{H}_2)$.

3. Recursively find expressions E_1 and E_2 that are sound outerjoin orderings for $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively.

4. Return the expression $E_1 \overset{\circ}{\bowtie} E_2$. □

Example 5.2: In Example 5.1 we learned that a γ-decomposition for our running example of Fig. 1.2 was $\mathcal{H}_1 = \{UDF, UDS\}$ and $\mathcal{H}_2 = \{UA\}$. This decomposition is based on selecting $X = U$ in the Bachman diagram of Fig. 3.3. The resulting Bachman diagrams for the components are shown in Fig. 5.4. Note that U is not needed in either.

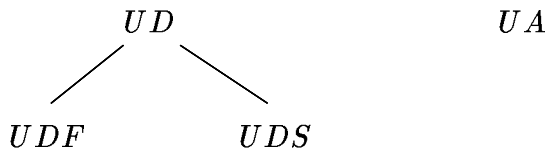

Fig. 5.4. Decomposing the Bachman diagram for Example 5.2.

Recursively, we can decompose $\mathcal{H}_1$ into $\{UDF\}$ and $\{UDS\}$. Thus, one sound outerjoin ordering is $UDF \overset{\circ}{\bowtie} UDS$ (for $\mathcal{H}_1$), and the other is UA (for $\mathcal{H}_2$). The sound outerjoin ordering for the entire hypergraph is $(UDF \overset{\circ}{\bowtie} UDS) \overset{\circ}{\bowtie} UA$. □

We may summarize the results of this and the preceding section by:

Example 5.3: Consider the hypergraph $\{AB, BC, CD\}$ of Example 3.3 and its Bachman diagram in Fig. 3.4. Here, we have a choice of minimal nodes — B or C. Let us choose B. When we delete node B from Fig. 3.4 we get the two Bachman diagrams of Fig. 5.5.

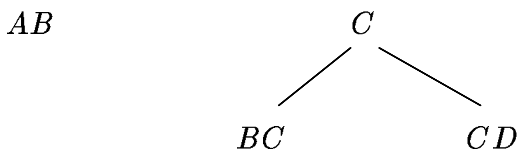

Fig. 5.5. Decomposing the Bachman diagram for Example 5.3.

Thus $\mathcal{H}_1 = \{AB\}$ and $\mathcal{H}_2 = \{BC, CD\}$. The decomposition of the former is a basis case; we just return expression AB. The latter is decomposed by deleting C and results in the sound outerjoin ordering $BC \overset{\circ}{\bowtie} CD$. Completing the recursion, we join these to get sound outerjoin ordering $AB \overset{\circ}{\bowtie} (BC \overset{\circ}{\bowtie} CD)$. Of course, had we decomposed on C first, we would get another sound outerjoin ordering, $(AB \overset{\circ}{\bowtie} BC) \overset{\circ}{\bowtie} CD$. □

Example 5.4: Let us modify Example 5.3 by adding B to the hypergraph; that is, $\mathcal{H} = \{AB, BC, CD, B\}$. The Bachman diagram is the same: Fig. 3.4, but now B is a "real" node rather than an intersection. That change affects how we decompose.

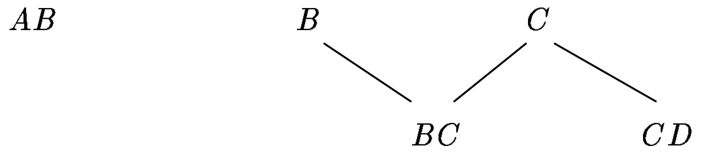

Fig. 5.6. Decomposing the Bachman diagram for Example 5.4.

We still have the choice of decomposing first on B or C; let us again choose B. $\mathcal{H}_1$ is still AB alone, but $\mathcal{H}_2$ also includes B. Thus, the two Bachman diagrams are as shown in Fig. 5.6. $\mathcal{H}_1$ is a basis case producing expression AB as in Example 5.3. However, $\mathcal{H}_2$ may still be decomposed in two ways: via B or C. Suppose we choose B again. Then Case 1 of Lemma 5.2 applies, and we let $\mathcal{H}_{21} = \{B\}$ and $\mathcal{H}_{22} = \{BC, CD\}$. BD($\mathcal{H}_{21}$) has only node B, while BD($\mathcal{H}_{22}$) has the nodes BC, CD, and C of Fig. 5.6.

We recursively compute sound outerjoin ordering $BC \overset{\circ}{\bowtie} CD$ for $\mathcal{H}_{22}$ and B for $\mathcal{H}_{21}$. These are joined to get the sound outerjoin ordering $B \overset{\circ}{\bowtie} (BC \overset{\circ}{\bowtie} CD)$ for $\mathcal{H}_2$. Last, we produce expression $AB \overset{\circ}{\bowtie} \big(B \overset{\circ}{\bowtie} (BC \overset{\circ}{\bowtie} CD)\big)$ for $\mathcal{H}$. This expression is one of several possible solutions. □

Theorem 5.1: Algorithm SOJO produces a sound outerjoin ordering for $\mathcal{H}$.

Proof: The proof is a simple induction on the number of hyperedges of $\mathcal{H}$.

BASIS: For the basis, one hyperedge, clearly the relation for this hyperedge is its full disjunction.

INDUCTION: Let $\mathcal{H}$ have $m > 1$ hyperedges and assume the theorem for smaller hypergraphs. Since every subhypergraph of a γ-acyclic hypergraph is a γ-acyclic hypergraph (because a γ-cycle of one part is a γ-cycle of the whole), we know $\mathcal{H}_1$ and $\mathcal{H}_2$ are smaller γ-acyclic hypergraphs. By the inductive hypothesis, both components have sound outerjoin orderings. By Lemma 5.1, if we apply the outerjoin operator to the expressions that are the two sound outerjoin orderings, the result is a sound outerjoin ordering for $\mathcal{H}$. □

Theorem 5.2: A hypergraph has a sound outerjoin ordering if and only if it is connected and γ-acyclic. Moreover, the sound outerjoin ordering can be obtained by γ-reduction.

Proof: From Theorems 4.1 and 5.1, plus the simple observation that a disconnected hypergraph cannot have a sound outerjoin ordering because of the Cartesian-product problem discussed at the beginning of Section V. □

VI. Summary

We have proven the following characterization of γ-acyclic hypergraphs:

- The connected γ-acyclic hypergraphs are exactly those hypergraphs for which we can compute the full disjunction by correctly ordering the outerjoins of the relations corresponding to its hyperedges (the sound outerjoin ordering).

There are also several interesting extensions of the results of this paper that we hope will appear in subsequent works:

- We have seen that for a γ-acyclic relation scheme, not every outerjoin ordering is a sound outerjoin ordering. The sound outerjoin for a relation scheme is not necessarily unique, however. Let us say that an outerjoin expression is *connected* if every subhypergraph corresponding to a subexpression is connected. It is clear that no disconnected outerjoin expression can possibly be sound, because it involves a cross product of some subset of the relations. However, we can characterize the subclass of the γ-acyclic hypergraphs for which every connected outerjoin expression is sound; it is those such that for no nonempty intersection of hyperedges is there a third edge that is not contained in the intersection but that contains a proper part of the intersection.
- There is a small generalization of γ-acyclic hypergraphs that characterizes those hypergraphs for which an outerjoin expression, followed by elimination of subsumed tuples, produces the full disjunction. An example is $\{AB, BC, ABC\}$, which, as discussed in Example 4.1, "almost" has a sound outerjoin ordering, but produces a relation with subsumed tuples that need to be eliminated in a subsequent phase.
- We are investigating techniques to compute efficiently the full disjunction when the hypergraph is not γ-acyclic. We are exploring both expressions and programs that combine outerjoins and outerunions. It is clear that the full disjunction of any relation scheme can be computed by an expression involving the above two operators; the question we wish to address is finding a minimum such expression.
- We can use our techniques to characterize those natural outerjoins that are associative.
- The algorithm described in this paper is being used in the "Information Manifold" project at Bell Laboratories to integrate facts gleaned from text search of HTML and SGML documents into relations.

Acknowledgements

We wish to thank Mihalis Yannakakis for suggesting the use of Bachman diagrams to simplify algorithm SOJO and the proof of its correctness. We also thank Dallan Quass for initial discussions regarding the problem of computing full disjunctions for tree schemes.

References

ANSI [1992]. Standard X3.135–1992, American National Standards Institute, New York.

Bernstein, P. A. and N. Goodman [1981]. "The power of natural semijoins," *SIAM J. Computing* **10**:4, pp. 751–771.

Fagin, R. [1983]. "Degrees of acyclicity for hypergraphs and relational database schemes," *J. ACM* **30**:3, pp. 514–550.

Fagin, R., A. O. Mendelzon, and J. D. Ullman [1982]. "A simplified universal relation assumption and its properties," *ACM Trans. on Database Systems* **7**:3, pp. 343–360.

Galindo-Legaria, C. [1994]. "Outerjoins as disjunctions," *ACM SIGMOD International Conf. on Management of Data*, pp. 348–358.

Graham. M. H. [1979]. "On the universal relation," technical report, Univ. of Toronto, Toronto, Ont., Canada.

Lien, Y. E. [1982]. "On the equivalence of database models," *J. ACM* **29**:2, pp. 333–363.

Maier, D., D. Rozenshtein, and D. S. Warren [1986]. "Window Functions," in Kanellakis, P. (ed.) *Advances in Computing Research* **3**, JAI Press, London, pp. 213–246.

Maier, D., J. D. Ullman, and M. Y. Vardi [1984]. "On the foundations of the universal relation model," *ACM Trans. on Database Systems* **9**:2, pp. 283–308.

Papakonstantinou Y., H. Garcia-Molina, and J. Widom [1995]. "Object exchange across heterogeneous information sources," Intl. Conf. on Data Engineering, Taipei, March, 1995. Available by anonymous ftp as `pub/papakonstantinou/1994/object-exchange-heterogeneous-is.ps` from `db.stanford.edu`.

Ullman, J. D. [1989]. *Principles of Database and Knowledge-Base Systems, Vol. II: The New Technologies*, Computer Science Press, New York.

Yannakakis, M. [1982]. "Algorithms for acyclic database schemes," *Proc. International Conference on Very Large Data Bases*, pp. 82–94.

Yu, C. T. and M. Z. Ozsoyoglu [1979]. "An algorithm for tree-query membership of a distributed query," *Proc. IEEE COMPSAC*, pp. 306–312.