On the Universal Relation

Marc H. Graham

University of Toronto

Toronto, Canada

1. Introduction

At the 1978 Very Large Data Base conference a paper was presented in which the wisdom of the "universal relation assumption" was questioned and more research was called for [BBG]. Since then it has been discovered that the test for the existence of an instance of the universal relation corresponding to a given database is intractable [HLY]. This paper reviews that work and then searches for schemas and databases for which efficient tests exist. The outline of the paper is as follows: After the basic notation is given in Section 2, section 3 reviews the work of [HLY]. Sections 4 through 8 investigate a particular test for join consistency and situations in which it may be applied. Section 9 briefly pursues a different course. Section 10 presents open questions for further research.

2. Basic Definitions and Notation

Very rarely in a discussion of databases is it necessary to introduce an infinite object. All the objects to be encountered in this paper are finite. In particular, that which we call the *universe*, and denote, U is a finite set of tokens which are themselves called *attributes*. In real world examples, attributes are such things as EMPLOYEE NUMBER, SALARY, SKILLS, ..., for a personnel applica-

tion or IDENTIFIER, DATATYPE, STORAGE LOCATION, ..., for a compiler's symbol table. As we are to investigate technical aspects of relational database theory, it is more convenient to use single letters, which we take from the beginning of the latin alphabet and often subscript: A_1, A_2, ..., B, C; as attributes. When necessary we will use capital letters towards the end of the latin alphabet: X, Y, ..., ; to denote subsets of the universe and following established custom we write set union as an operator free expression and deliberately confuse a single attribute and the set containing only that attribute. Thus A_jX should be read as $\{A_j\} \cup X$.

We now introduce data values which are things such as **12345** and **real.** Our first step is to associate with each attribute a set of values called a *domain.* For EMPLOYEE NUMBER we might use {"strings of length 5 over 0,1, ..., 9"}; for DATATYPE: {**real, integer, logical, string, ...**}. Usually distinct attributes are allowed to have either distinct or identical domains. We will be drawing no consequences from the nature of the domains; therefore, we are free to assume all attributes share a single domain, denoted D. In our subsequent examples, D need be no larger than {0, 1, 2}, but our results hold for arbitrary assignments of finite domains.

Attributes are assigned attribute values in groups. The object which performs this assignment is called a *tuple*. A tuple is a function from the universe to the domain. In symbols:

$$t: U \rightarrow D$$

We use the letters: t, u, v, ..., occasionally subscripted, to denote tuples.

A tuple serves to identify and relate attribute values. So we might have a tuple, t, with t(EMPLOYEE NUMBER) = **12345**, t(SALARY) = **20000**, t(IDENTIFIER) = **"IDENTIFIER"**, t(DATATYPE) = **string**. It is easier to display tuples as vectors or "ordered n-tuples" such as **(12345, 20000, "IDENTIFIER", string)** by arbitrarily deciding upon some ordering of the attributes. (Thus the name, tuple.) This

makes the tuples appear to be elements of the Cartesian product of the underlying domain taken with itself and this was the original definition.

A tuple need not be defined on every attribute in the universe. Let the set of attributes on which a given tuple, t, is defined be called its *scheme* and denoted $\alpha(t)$. A set of tuples is uniform if each tuple has the same scheme. A ***relation*** is a uniform set of tuples. A relation's scheme (or *schema*) is the scheme of its tuples. Relations are conventionally denoted by subscripted lowercase r's: $r_1, r_2, \ldots, r_i, \ldots$, occasionally s; relation schemes by subscripted upper case R's: $R_1, R_2, \ldots, R_i, \ldots$, occasionally S; obeying the equation $\alpha(r_i)=R_i$. ($\alpha(r_i)$ is just $\alpha(t)$ for t an element of r_i, which is fine whenever r_i has any elements. The empty set is certainly uniform and thus a relation but we do not know what schema to give it. We will blithely assume α to be well defined everywhere and that in particular when confronted with an empty relation it can distinguish which empty relation it is.)

Relations may be displayed as tables or matrices in which the columns represent attributes, each column being labelled by the attribute it represents, and the rows are tuples in the vector display mode already mentioned. There are examples of such displays in figures 1 thru 3.

When for some r_j we have $\alpha(r_j)=U$, then we say r_j is an *instance of the universal relation* or merely an instance. An instance is often denoted by an upper case I. The phrase "the universal relation" reflects the customary misuse of the term "relation" for "relation schema." We will continue this custom, relying on context for disambiguation.

A *database* is a set of relations. These are denoted by an upper-case bold **R** as in $\mathbf{R}=\{r_1, \ldots, r_k\}$ for a database of k relations. A *database schema* is a set of relation schemas and not surprisingly we write

$$\alpha(\mathbf{R})=\{\alpha(r_1), \ldots, \alpha(r_k)\}=\{R_1, \ldots, R_k\}=R.$$

It is easier to ignore the empty database than it was to ignore the empty relation. We also safely ignore databases in which distinct relations have the same schema.

We will be concerned with two operators of the relational algebra, *projection* and *join*. Projection forms its result relation by ignoring some of the attributes of its operand relation. Let t be a tuple and let X be a subset of $\alpha(t)$. The projection of t onto X, written $t.X$, is a tuple, u, with the properties that $\alpha(u)=X$ and for each attribute $A \in X$, $t(A)=u(A)$. The projection of a relation onto a subset of its attributes is the set formed by the projections of its tuples onto that set of attributes. Two or more tuples having the same projection are identified in the projection of the relation. For r a relation and $X \subseteq \alpha(r)$, we write projection as $\pi_X(r)$ and define it as

$$\pi_X(r)=\{u \mid \exists\, t \in r \wedge u=t.X\}$$

We use different notations for projection of tuples and projection of relations because the two operations behave differently. We can illustrate this difference with the following rule which is obviously sound.

Subset Rule for Projection: Let r_1, r_2 be relations. Let $X \subseteq \alpha(r_1) \cap \alpha(r_2)$. If $\pi_X(r_1)=\pi_X(r_2)$ then for every proper subset Y of X, $\pi_Y(r_1)=\pi_Y(r_2)$.

The same rule holds for projections of tuples. Somewhat surprisingly, the converse of the subset rule is false for relations even though it is true for tuples. Figure 1 is a counterexample. We leave it to the reader to convince himself that the projections of the relations in that figure on any subset of their attributes are equal.

$U=\{A,\ B,\ C\}$
$D=\{1,\ 2\}$

r_1

A	B	C
1	1	1
2	1	1
1	2	1
1	1	2

r_2

A	B	C
2	1	1
1	2	1
1	1	2

Figure 1
A counterexample to the
converse of the subset rule

The *natural join* takes two relations as operands and forms a new relation whose schema is the union of the operand schemata. For relations r_1, r_2 with schemas $\alpha(r_1)=R_1$, $\alpha(r_2)=R_2$, we define $r_1 * r_2$ by

$$r_1 * r_2 \equiv \{t \mid (\exists t_1 \in r_1)(\exists t_2 \in r_2)(t.R_1=t_1) \wedge (t.R_2=t_2)\}$$

We can calculate $r_1 * r_2$ by examining the tuples that appear in the projections of r_1 and r_2 onto their common attributes; i.e., the relations $s_1=\pi_{\alpha(r_1)\cap\alpha(r_2)}(r_1)$ and $s_2=\pi_{\alpha(r_1)\cap\alpha(r_2)}(r_2)$ and finding the tuples which appear in both, namely $s_3=s_1\cap s_2$. For each tuple $u \in s_3$ we find the tuples in each of r_1 and r_2 which have u as their common attribute projection and form a tuple of $r_1 * r_2$ for every pair. (This is not meant to be an efficient calculation technique but only to help explain the definition.) Therefore any tuple of r_1 (or r_2) whose projection onto $\alpha(r_1)\cap\alpha(r_2)$ does not match any tuple of r_2 (or r_1) is lost from the join. Conversely any tuple which does match appears indiscriminately with all the matching tuples of the other relation. In the special case of $\alpha(r_1)\cap\alpha(r_2)=\phi$, the join is exactly the cross-product: every tuple of r_1 appears with every tuple of r_2. In the case $\alpha(r_1)\subseteq\alpha(r_2)$, the join is the intersection: those tuples of r_2 whose $\alpha(r_1)$ projection appears in r_1.

It is not difficult to see that join is an associative and commutative operator; in short, that $(r_1 * r_2) * r_3 = r_1 * (r_2 * r_3)$ and $r_1 * r_2 = r_2 * r_1$. We are justified in

writing the expression representing the join of a group of k relations as

$$r_1 * r_2 * \cdots * r_k$$

meaning the result of any one of the implied sequences of binary joins. For convenience we abbreviate this to $\mathop{*}_{i=1}^{k} r_i$ or, when joining all the relations of a database $\mathbf{R}=\{r_1, r_2, \ldots, r_k\}$, to $*\mathbf{R}$. It can easily be shown, by induction on k, that, for $\alpha(r_i)=R_i$,

$$*\mathbf{R}=\{t \mid \bigwedge_{i=1}^{k} (\exists t_i \in r_i \wedge t.R_i=t_i)\}$$

3. The General Case

We now begin our formal investigation into the difficulty of enforcing the universal relation assumption as a database constraint. This section summarizes the negative results of [HLY] and [L].

A pair of tuples selected from distinct relations of a database are *compatible* if they agree on the attributes on which both are defined. Thus t, w are compatible if $t.(\alpha(t) \cap \alpha(w))=w.(\alpha(t) \cap \alpha(w))$.

Proposition 1. (The Compatible Tuples Condition). Let $\mathbf{R}=\{r_1, \ldots, r_k\}$ be a database. Select r_j from $\mathbf{R}$ and u from r_j. An instance of the univeral relation exists for $\mathbf{R}$ if and only if a set of tuples, $\{v_1, \ldots, v_k\}$ can be found such that for all $1 \leq i, l \leq k$

- $v_i \in r_i$
- $v_j = u$
- v_i and v_l are compatible

Proof. The necessity is apparent: u is the projection onto $\alpha(r_j)$ of some tuple of I. The v_i are the projections of that tuple onto the remaining relation schemes. Conversely, the v_i build a tuple of I whose projection onto $\alpha(r_j)$ is u and such a tuple can be built for each tuple of each relation in $\mathbf{R}$. ∎

Direct verification of the compatible tuples condition leads to a "backtracking" algorithm. However, the next proposition seems to offer some hope.

Proposition 2. Let **R** be a database satisfying the compatible tuples condition and let I be any corresponding instance. Then we have

1) $I \subseteq {}^*\mathbf{R}$

2) $\pi_{\alpha(r)}({}^*\mathbf{R}) = r$ for every $r \in \mathbf{R}$

Proof. For part 1 see [ABU]. For part 2 see [HLY]. ∎

In [HLY] a database which satisfies the universal relation assumption is called *join consistent*. Proposition 2 gives a justification for this name. It states that the join of a join consistent database is its largest instance. This implies the correctness of a simple algorithm to test join consistency: form the join and test the projections. Regrettably, this algorithm has worst case behaviour $O(m^k)$ (k the number of relations, m the size of each relation). It may be deduced from the next theorem that no algorithm with running time a polynomial of fixed degree is likely to be found to solve this problem in general.

Theorem 1 Determining join-consistency is NP-complete.

Proof. [HLY] A reduction from graph vertex 3-colorability is given. ∎

Note Readers unfamiliar with the notion of NP-completeness are referred to [GJ]. For our purposes it will suffice to assert than an NP-complete problem is probably too difficult to be solved in a reasonable amount of time by any algorithm.

Knowing that a database is join-consistent does not seem to help in determining if a modified database will be join-consistent. This is proven in [HLY] for the case that the modification is a tuple insert.

4. Tractable Subproblems

Since it appears that the problem of determining join consistency is intractable in general, we turn our attention to subproblems which can be shown to have efficient algorithms. In particular, we will give characterizations of subproblems for which it is sufficient to test the following condition, which is a weakened form of the compatible tuples condition.

Definition A database $R=\{r_1, \ldots, r_k\}$ satisfies the *common intersection property (CIP)* if for all $r_i, r_j \in R$, $X=\alpha(r_i)\cap\alpha(r_j)$, $\pi_X(r_i)=\pi_X(r_j)$. That is, every pair of relations agree on their common attributes. [Z, chap 5.]

The test for CIP is polynomial-time bounded for any database. As the compatible tuple condition can be shown to imply the common intersection property, CIP will hold in any join-consistent database. The insufficiency of CIP can be demonstrated by the example in figure 3, which also appears in [HLY]. The reader may verify that the database is not join consistent by forming the join. We will formalize what "goes wrong" with this example in a later section.

U = {A, B, C}

r_1

A	B
1	1
2	2

r_2

B	C
1	1
2	2

r_3

C	A
1	2
2	1

Figure 3

So far we have considered properties defined on the data in a database. We now seek schema properties which characterize databases in which CIP implies join-consistency. Specifically we examine the pattern of intersection of the relation schemes. Let $R=\{R_1, \ldots, R_k\}$ be a database schema over a universe U. An attribute $A \in R_j$ is said to be *common* if it appears in some intersection $R_j \cap R_k$ for some $k \neq j$. As we will be studying intersections, we can ignore any

attribute which is not common. Consider, in justification, adding any set of attributes, X, to R_1 in figure 1. No way of assigning values to X will affect the conclusion that no universal instance exists. Now assume that for some distinct schemes R_j, R_k, we have $R_j \subseteq R_k$. Then we are safe in studying the schema $R-\{R_j\}$, in the sense that any database satisfying CIP and having schema R will have a universal instance just in case the database resulting from the removal of a relation r_j with schema R_j has a universal instance. These observations lead to the following algorithm, which outputs "yes" only if its input is a schema for databases in which CIP implies join-consistency.

Algorithm 1

Input: A schema $R=\{R_1, \ldots, R_k\}$ on universe U.

Ouput: "yes" or "no"

Procedure:

Step1: For each i from 1 to k remove from R_i any non-common attributes. If R_i becomes empty remove it from R.

Step2: Find, if possible, R_i, R_j in R with $R_i \subseteq R_j$. Remove R_i from R.

Step 3: Repeat steps 1 and 2 until no changes are made. If R is empty output "yes"; otherwise output "no". ∎

If the number of relations in R is k and the number of attributes in U is n, then Step1 can be done in time $O(kn)$; step2 in time $O(k^2n)$; step3 can cause at most k iterations. Therefore algorithm 1 is $O(k^3n)$.

5. A necessary condition

We now apply some techniques of graph theory to our problem. It is assumed the reader is familiar with the elementary aspects of graph theory as may be found in [H].

For a given family of subsets of a given set, an *intersection graph* is a graph is which the subsets play the role of vertices and an edge connects pairs of subsets whose intersection is non-empty. A *common attribute graph (CAG)* is a labelled intersection graph in which the label on an edge is the intersection giving rise to the edge and the label on a vertex is the union of the labels on the edges incident to the vertex. Formally, if R is a database schema on U, then $CAG(R)=(V,E,l)$ where

- $V=R$
- $E=\{(R_i,R_j) \mid R_i \cap R_j \neq \phi\}$
- $l: V \cup E \rightarrow P(U)$

where $P(U)$ is the set of all subsets of U and l is defined by

$$l(x)=\begin{cases} R_i \cap R_j & \text{if } x \in E \text{ and } x=(R_i,R_j) \\ \bigcup_y l((x,y)) & \text{if } x \in V \text{ and } (x,y) \in E \end{cases}$$

This choice of vertex labelling automatically removes non-common attributes from our attention. There are examples of CAG graphs in figures 4 and 5.

We can rapidly establish some partial results by considering graphs in which the edge labels are pairwise disjoint.

Proposition 3 Let the edge labels of $CAG(R)$ be pairwise disjoint. Then R is a scheme in which CIP implies join consistency if and only if $CAG(R)$ is acyclic.

Proof

If It is easy to see that algorithm 1 will output "yes" for any acyclic graph. In particular, the set of common attributes of any leaf in such a graph is exactly the label on the edge connecting the leaf to the graph.

Only if

We construct a counterexample. Let C be any cycle and R_i any vertex of C. Construct a relation for each vertex of $CAG(R)$ other than R_i containing two

tuples: one a vector of all 1's, one a vector of all 2's. Let e be an edge of C incident with R_i. Construct a two tuple relation for R_i as follows: one tuple contains all 1's except for attributes in $l(e)$ which are assigned 2's. The other tuple has 2's everywhere except for 1's in $l(e)$. This construction satisfies CIP by virtue of the fact that no attribute appears in more than one edge label.

Now $\mathop{*}_{j \neq i} R_j$ is clearly a relation on all the attributes of the universe and having two tuples: one of all 1's and one of all 2's. Just as clearly, $(\mathop{*}_{j \neq i} R_j) * R_i = \phi$. Thus the constructed database state satisfies CIP without being join consistent, as required. ∎

Let R be a schema with $CAG(R)$ as in the statement of proposition 3. Let W be a set of attributes not appearing in R. Consider enlarging the schemas of some subset of R by adding W to each relation schema in the subset. Let the new database schema be R'. $CAG(R')$ will in general no longer have all pairwise disjoint edge labels; however, if $CAG(R)$ contains cycles, the proof of proposition 3 will go through for R' if the attributes in W are uniformally assigned the value 0. We have established the following

Corollary For any schema, R, let R' be the result of removing from R any attribute appearing in three or more relations (equivalently, two or more edge labels). R is a schema for which CIP implies join consistency only if $CAG(R')$ is acyclic . ∎

6. A complete solution for databases on three relations

We are able to characterize all databases with three relations for which CIP implies join consistency. (It is obvious that CIP implies join consistency for all databases on two relations.) In the process we introduce a condition which will later be seen to be sufficient in the general case.

Definition We say that a CAG, G, satisfies the *CAG-C condition* if every cycle of G contains two edges, say e_i, e_j with $l(e_i) \supseteq l(e_j)$. e_i, e_j are called *comparable edges*. G is called a *CAG-C graph*.

A graph in which every cycle of length 4 or more has a chord is called *triangulated* or *chordal*. It is easy to see that any CAG-C graph is chordal. The triangles of CAG graphs have some useful properties which are summarized in the next proposition.

Proposition 4 The Triangle Lemma

(1) In any CAG triangle, the intersection of any two edge labels is contained in the third edge label.

(2) In any CAG-C triangle, the intersection of any two edge labels is non-empty.

(3) If two edges of a CAG-C triangle are incomparable, the third edge label is their intersection.

Proof

(1) The intersection of two edge labels is the intersection of all three relations which is certainly contained in the intersection of the two relations forming the third edge.

(2) Assume otherwise. By the CAG-C property, the third edge is comparable to one of the two edges whose intersection is empty. By (1), the intersection of the two comparable edges, i.e., the smaller of the two, is contained in the remaining edge, violating the assumption.

(3) If the third edge contained one of the incomparable edges, (1) would be violated. Therefore the third edge is a lower bound on the two incomparable edges and must be the greatest lower bound by (1) which is not the empty set by (2). ∎

Figures 4 and 5 give the only possible CAG triangles for CAG-C and non-CAG-

C triangles respectively. All letters in these figures represent non-empty, disjoint sets of attributes except where noted. From the triangle lemma and previous considerations, we have

Theorem 2 The Triangle Theorem

If **R** is a database of three relations with schema R, then CIP is equivalent to join consistency for **R** iff one of the relation schemes of R contains all of the common attributes of R.

Proof Figure 4 gives the case for $CAG(R)$ a triangle. Inspection of the other graphs on three points completes the proof.∎

7. A technical result

The importance of the triangle theorem is that it shows that database schemas on three or fewer relations are fully understood with regard to this problem. The same is not true for larger schemata. The next result is a technical one about CAG-C graphs. Having it will allow us to discuss related work in this area.

A complete graph is one in which an edge connects each pair of vertices. A triangle is the complete graph on 3 vertices. We now prove a result about complete CAG-C graphs. A *minimal* edge of a CAG graph is one whose label is minimal among the set of all edge labels in the graph. A *maximal* edge is the dual notion. A *smallest* edge has a label contained in every edge label of the graph. The *largest* edge is also defined. Every non-trivial CAG graph has a set of minmal (maximal) edges but may or may not have a smallest (largest) edge.

Proposition 5 Let R be a database schema with $CAG(R)$ satisfying CAG-C. Then $CAG(R)$ is complete iff there is a non-empty set of attributes which appear in every relation of R.

Proof The if part is obvious. The only-if part is obvious if R contains fewer than 3 relations. For larger schemas, we prove the stronger statement given next.

Induction Hypothesis Let $R=\{R_1, \ldots, R_n\}$, $(n \geq 3)$ and assume $CAG(R)$ is complete. Let $S=\{S_1, \ldots, S_n\}$ be the collection of all subsets of R having $n-1$ elements. Then for all but at most two values for j, $1 \leq j \leq n$

$$\bigcap_{i=1}^{n} R_i = \bigcap_{k=1}^{n-1} R_{j_k}$$

(where $S_j=\{R_{j_1}, \ldots, R_{j_{n-1}}\}$).

Basis The triangle lemma.

Induction Assume the hypothesis has been proven for $n \leq k$. Consider a schema, R, on $k+1$ relations with $CAG(R)$ a complete CAG-C graph. Choose a minimal edge of $CAG(R)$. There are $\binom{k-1}{k-2}=k-1$ subsets of size k which contain both end points of this edge. By the induction hypothesis this edge is smallest for each subgraph induced by these subsets. But the subsets cover all the relations in R and all the edges in $CAG(R)$, establishing the induction hypothesis for $k+1$. ∎

Note that this proposition applies to any complete CAG graph each of whose triangles are CAG-C.

We have introduced proposition 5 in order to comment on the work of Zaniolo in chapter 5 of [Z]. That work investigates this problem with the help of hypergraphs and in particular the representative graph of a hypergraph. (See [Be] for definitions of hypergraphs.) These representative graphs are CAG graphs and what are called connection sets in [Z] are CAG edge labels. Zaniolo shows our result for "simply-connected hypergraphs" which in our terms are defined as follows: A CAG graph satifies the $CAG-Z$ property if in each cycle of the graph there is an edge whose label is contained in all other labels of the cycle. It can be shown, as a consequence of proposition 5, that CAG-Z characterizes CAG-C graphs each of whose blocks is complete. Therefore this work extends the result to more schemas.

8. CAG-C is sufficient

Before proceeding to the result of this section, we need the next lemma.

Lemma 6 A graph is CAG-C if and only if each cycle of the graph contains a pair of adjacent, comparable edges.

Proof The if part is immediate. The only-if part is to be found in the appendix. ∎

We are now nearly fully armed for our final assault of this section. Our main weapon is a transformation which operates simultaneously on a database and its schema.

Let **R** be a database satisfying CIP, $\alpha(\mathbf{R})=R$ the schema for **R**, and assume $CAG(R)$ = G satisfies CAG-C. Let L be a maximal label of G. Define a transformation, $T_L(\mathbf{R}, G)=(T_L(\mathbf{R}), T_L(G))$ as follows: Collect all the edges of G having L as a label and denote the resulting set of edges E_L. Let V_L be the set of endpoints (relation schemas) of the edges in E_L and let $\mathbf{R}_L$ be the relations in **R** with schemas in V_L. Form $T_L(\mathbf{R})$ by replacing $\mathbf{R}_L$ by $*\mathbf{R}_L$, the join of the relations incident to edges labelled by L. Define $T_L(G)$ as follows: Let $S=\bigcup V_L$ be a relation schema containing all the attributes in all the schemata of V_L. Form the vertex set of $T_L(G)$ by removing V_L and adding S. Form the edge set of $T_L(G)$ by removing E_L and replacing the edges between a vertex, R_i, not in V_L and any vertex in V_L with an edge (R_i,S). The edges not incident to vertices in V_L appear unchanged.

We define the labelling function in $T_L(G)$, denoted $l_{T_L(G)}$, in terms of l_G, the label function in G. It suffices to define its behaviour on edges: An edge of $T_L(G)$ which appears in G has the label it had in G. Otherwise, a new edge's label is the union of the labels on the edges it replaced. This is formalized by

$$l_{T_L(G)}(e) = \begin{cases} l_G(e) & e \text{ an edge of } G \\ \bigcup_{R_j \in V_L} l_G((R_i,R_j)) & e=(R_i,S)\wedge(R_i,R_j) \text{ an edge of } G \end{cases}$$

Note that if the relation S already appears in R, $T_L(G)$ is an induced subgraph of

G. It is easy to see that $T_L(G)$ is the CAG graph of the schema of $T_L(\mathbf{R})$.

A key feature of this transformation is that the edges of E_L form a complete subgraph of G on the vertices V_L. In order to see this, consider two edges, e, f of E_L with $e=(R_{e1},R_{e2}), f=(R_{f1},R_{f2})$. Then all four relations contain all the attributes in L. Therefore the edge (R_{e1},R_{f1}), if it exists (it will not exist precisely when e and f are incident with $R_{e1}=R_{f1}$), has a label which contains L. But since L is maximal in G, the containment must be improper.

We will need to prove three propositions concerning this transformation. The first is relatively simple.

Proposition 7 An instance of $T_L(\mathbf{R})$, if it exists, is an instance of $\mathbf{R}$.

Proof Since the relation schemes in V_L each share exactly the attributes in L and since $\mathbf{R}$ satisfies CIP, the compatible tuples condition holds amongst the relations in $\mathbf{R}_L$ and their join is an instance for them, by propositions 1 and 2. ▪

Before proceeding we need to make an observation. Let R_i be a vertex of G in V-V_L which is adjacent to some vertex in V_L. R_i appears in $T_L(G)$ adjacent to S. Let $l_{T_L(G)}((R_i,S))=L_1X$, where $L_1 \subseteq L$ and X is disjoint from L and non-empty. (If no candidate for R_i can be found with these properties, we do not need the observation.)

Observation There is exactly one $R_j \in V_L$ such that $X \subseteq R_j$.

Proof Clearly there cannot be two such relations since $X \not\subseteq L$. It remains to show there is at least one. Let $X=AB\cdots$ and assume the existence of two distinct vertices of V_L, R_k, R_l with $A \in R_k$, $B \in R_l$. Then the edges (R_i,R_k) and (R_i,R_l) are incomparable in G and by the triangle lemma, part 3, $l_G((R_k,R_l))=L$ is their intersection which is impossible since L is maximal in G. ▪

As a consequence of this observation we have that if an edge (R_i,S) in $T_L(G)$ has a label containing an attribute not in L, there is exactly one edge incident to

R_i in G having this label. (The case of S appearing in G is special; however, the statement holds then as well.) Otherwise all edges between R_i and the vertices of V_L have the same label in G; namely, $l_{T_L(G)}((R_i,S))$. The fact that the edges added to G to form $T_L(G)$ do not have "fat" labels is crucial in the remainder of the development.

Proposition 8 $T_L(\mathbf{R})$ satisfies CIP.

Proof Edges not incident to S represent parts of the CIP constraint not affected by the transformation. The remainder of the proof follows from proposition 7, the observation, and the subset rule for projections. ∎

Proposition 9 $T_L(G)$ satisifies CAG-C.

Proof Let C be a cycle of $T_L(G)$ which violates CAG-C. Each edge of C not incident to S has the same label as the equivalent edge in G. Let the edges (R_k,S), (R_j,S), in C have incomparable labels L_1X, L_2Y, respectively, with X and Y disjoint from L. By the observation we can choose R_k, R_l in V_L such that $l_G((R_i,R_k))=L_1X$ and $l_G((R_j,R_l))=L_2Y$. If X and Y are comparable then we have $R_k=R_l$. Thus C is a non-CAG-C cycle of G. If X and Y are incomparable, $R_k \neq R_l$ and the sequence of edges (R_i,R_k), (R_k,R_l), (R_l,R_j) have incomparable labels. Therefore G has a cycle without adjacent, comparable edges and is not CAG-C by lemma 6. ∎

Theorem 3 If **R** is a database satisfying CIP, R is the schema of **R** and $CAG(R)$ satisfies CAG-C, then **R** is join consistent.

Proof Continue to apply transformations of the type described to **R** and $CAG(R)$ until a state is reached in which no further transformation can be made. These transformations generate a sequence of database, schema pairs: $\{<\mathbf{R}_0,R_0>, <\mathbf{R}_1,R_1>, \ldots, <\mathbf{R}_n,R_n>\}$ with $\mathbf{R}_0=\mathbf{R}$, $R_0=R$ and each pair in the sequence a transformation of the previous pair. Since each transformation results in a smaller schema, all such transformation sequences must terminate. Since no transformation can be applied to $CAG(R_n)$ it must be the case that $CAG(R_n)$ has

no edges. Thus $\mathbf{R}_n$ consists of a set of relations having no common attributes. Such a database is always join consistent. Since each $<\mathbf{R}_i, R_i>$ pair represent a CAG-C schema and CIP database by propositions 8 and 9, proposition 7 allows us to deduce the join consistency of **R**. ▪

9. Yet another sufficient condition

This next, somewhat bizarre result shows that there is at least some connection between join consistency and dependency theory.

A *join dependency* is written $*[S]$, where S is a set of sets of attributes. A relation r all of whose attributes are in S satisfies $*[S]$ if r is a fixed point of the "projection-join mapping" associated with S [BMSU]. If $S=\{S_1, \ldots, S_k\}$, then r satisfies $*[S]$ iff

$$\mathop{*}_{i=1}^{k} (\pi_{S_i}(r)) \doteq r$$

See [R2] [F2] for more on join dependencies.

Proposition 10 Let **R** be a database satisfying CIP and having schema R. If each relation in **R** satisfies the join dependency defined by labels of edges incident to it in *CAG(R)*, then **R** is join consistent.

Proof. We show that the compatible tuples condition is satisfied. Assume for some tuple t_1 in relation r_1 that the condition fails to be satisfied. Let $S=\{t_1, \ldots, t_k\}$ be a maximal set of tuples (with t_i from relation r_i: no more than one tuple from any relation) within which the requirements of compatibility are satisfied. We know that relation r_{k+1} (from which no tuple appears in S) has tuples $w_1, \ldots, w_k$, not necessarily distinct, such that $w_i.(\alpha(r_i) \cap \alpha(r_{k+1})) = t_i.(\alpha(r_i) \cap \alpha(r_{k+1}))$. Now these projections are join consistent, therefore their join appears in (the appropriate projection of) r_{k+1}, by the join dependency. But this tuple may be added to S, contradicting its maximality. ▪

The proof shows that an exhaustive search algorithm for the compatible tuples condition succeeds in such a database without ever having to backtrack. This is hardly necessary for join consistency.

10. Open questions

There are many questions left open by this work. No schema condition both necessary and sufficient for CIP to imply join consistency has been uncovered. It is not known whether or not all CAG-C graphs are accepted by Algorithm 1. There are some schemas acceptable to Algorithm 1 whose graphs are not CAG-C. The condition of section 9 is independent of the other conditions described. It restricts the data rather than the schema. It is much too restrictive. Prehaps a more useful contribution of dependency theory can be found.

Do any of the known schema design techniques guarantee an easy join consistency test for their designs? Note that the schema of example 3 is the output of [BDB] given $\{A \rightarrow C,\ B \rightarrow C\}$ as input. Furthermore it is isomorphic to the example of [Ni]. Although current design criteria (independent components in normal form) do not produce schemas in which CIP implies join consistency, this does not mean that they may not have easy tests.

Even if the class of schemas with polynomial time bounded join consistency tests is very large, one may argue on semantic grounds that the universal relation assumption over constrains the database. It requires every value of every attribute to have some relation to some value of every other attribute in the universe. This is frequently not the case in practice: a department in the planning stage has no employees. An important goal of current research must be to weaken the universal relation assumption to deal with these difficulties without losing the advantages the assumption provides.

References

[A] Armstrong, W.W.; "Dependency structures of database relationships," *Proc 1974 IFIP Congress*, pp 580-583, 1974

[ABU] Aho,A.V.;Beeri,C.;Ullman,J.; "The Theory of Joins in Relational Databases," *TODS 4:3*, pp 297-314, 1979

[B] Bernstein,P.A.; "Synthesizing Third Normal Form Relations from Functional Dependencies," *TODS 1:4*, pp 277-298, 1976

[BB] Beeri,C;Bernstein,P.A.; "Computational Problems Related to the Design of Normal Form Relational Schemas," *TODS 4:1*, pp 30-59, 1979

[BBG] Beeri,C;Bernstein,P.A.;Goodman,N; "A Sophisticate's Introduction to Database Normalization Theory," *Proc of 4th VLDB Conf.*, pp 113-124, 1978

[BDB] Biskup,J.;Dayal,U.;Bernstein,P.A.; "Synthesizing Independent Database Schemas," *Proc. SIGMOD Conf.*, pp 143-151, 1979

[Be] Berge,C.; *Graphs and Hypergraphs*, North-Holland, 1973

[BFH] Bernstein,P.A.;Fagin,R.;Howard,J.H.; "A complete axiomatizaton for functional and multivalued dependencies," *Proc SIGMOD Conf*, pp47-61, 1977

[BG] Bernstein,P.A.;Goodman,N.; "What does Boyce-Codd Normal Form do?" *TR-07-79* Harvard University

[BMSU] Beeri,C;Mendelzon,A.O.;Sagiv,Y;Ullman,J.D.; "Equivalence of Relational Database Schemes," *Proc. 11th ACM Symp. on Theory of Comp.*, pp 319-329, 1979

[C] Codd,E.F.; "Further Normalization of the Database Relational Model," *Data Base Systems*, Prentice-Hall, pp 33-64, 1972

[F1]
[F2]
[F3]
[GJ]
[H]
[HLY]
[L]
[Ni]
[R1]
[R2]
[Z]

[F1] Fagin, R; "Multivalued dependencies and a new normal form for relational databases," *TODS 3:3*, pp 201-222, 1978

[F2] Fagin,R; "Normal forms and relational database operators," *Proc SIGMOD Conf.*, pp 153-160, 1979

[F3] Fagin,R; *A normal form for relational databases that is based on domains and keys*, IBM RJ2520, May 1979

[GJ] Garey,M.R.;Johnson,D.S.; *Computers and Intractability*, W.H.Freeman, 1979

[H] Harary,F; *Graph Theory*, Addison-Wesley, 1972

[HLY] Honeyman,P.;Ladner,R.E.;Yannakakis,M; "Testing the Universal Instance Assumption," *IPL* 10:1 Feb 80

[L] Ladner,R.E.; private communication

[Ni] Nicolas,J.M.; "Mutual dependencies and some results on undecomposable relations," *Proc. VLDB Conf.*, pp 360-367, 1978

[R1] Rissanen,J; "Independent components of relations," *TODS 2:4*, pp 317-325, 1977

[R2] Rissanen,J; "Theory of Relations for Databases - A Tutorial Survey," *Proc 7th Symp. on Math. Found. of Comp. Sci.*, Springer-Verlag, pp 536-551, 1978

[Z] Zaniolo,C; *Analysis and Design of Relational Schemata for Database Systems*, UCLA-ENG-7669, 1976

Appendix

The following notation is used in the proof of the next result. If e, f are edges in a CAG graph, we write $<e,f>$ to mean $l(e) \supseteq l(f)$ and $[e,f]$ to mean e and f are comparable; that is, either $<e,f>$ or $<f,e>$ holds. We use the symbol $\neg$ to mean negation.

Lemma 6 (Only-If Part) A CAG graph containing a cycle in which no adjacent pair of edges is comparable is not CAG-C.

Proof By contradiction. Let $G = CAG(R)$ for some schema R. Choose $C=R_0, R_1, \ldots, R_{k-1}$, a cycle of G and denote the edge (R_{i-1}, R_i) as e_i. (All index arithmetic is done mod k.) Assume C is chosen as a shortest cycle of G satisfying

$$\neg[e_i, e_{i+1}] \text{ for all } 0 \leq i \leq k-1 \qquad (*)$$

Since G is a CAG-C graph, we may assume, without loss of generality, the existence of i and j such that $<e_i, e_j>$ and for all k_1, k_2 with $0 \leq i \leq k_1 < k_2 \leq j \leq k-1$

$$\neg[e_{k_1}, e_{k_2}] \textit{ unless } k_1 = i \text{ and } k_2 = j \qquad (**)$$

In the set of edges from e_i to e_j inclusive, e_i, e_j is the only comparable pair. It is easy to see that $j > i+2$. Obviously, by (*) $j \geq i+2$. However $j = i+2$ allows the following computation

$$\begin{aligned} <e_i, e_{i+2}> &\rightarrow (R_{i-1} \cap R_i) \supseteq (R_{i+1} \cap R_{i+2}) \\ &\rightarrow (R_i \cap R_{i+1}) \supseteq (R_{i-1} \cap R_i \cap R_{i+1}) \supseteq (R_{i+1} \cap R_{i+2}) \\ &\rightarrow <e_{i+1}, e_{i+2}> \end{aligned}$$

which contradicts (*).

The attack for the remainder of this proof is to choose a sequence of chords of C which subtend nested intervals of the path from R_{i-1} to R_j. The innermost chord forms a triangle with two edges of C. The label of this chord is necessarily contained in both these edges, by the triangle lemma. It is then shown that the label on each chord of the sequence contains the label of each chord which precedes it. This leads to a contradiction.

The elements of the sequence are denoted by subscripted q's. Let $q_0=(R_{i-1},R_j)$. Let $q_{m-1}=(R_{k_1},R_{k_2})$. Then

$$[q_{m-1},e_{k_1+1}] \rightarrow q_m=(R_{k_1+1},R_{k_2}) \qquad (***)$$
$$[q_{m-1},e_{k_2}] \rightarrow q_m=(R_{k_1},R_{k_2-1})$$

If both antecedents hold,

the choice is arbitrary.

(see figure A.1)

The last element of the sequence is q_p for $p=j-i-2$.

We must now show that each of $q_0,q_1,\ldots,q_p$ is a well specified edge of G. This can be done by induction. q_0 is an edge of G by $<e_i,e_j>$. It suffices to show that at least one of the two conditions of (***) holds for $1\leq m\leq p$. Assume this is true for all $m\leq m_0$. Then q_{m_0} is an edge of G. But q_{m_0} and the edges of C which it subtends form a cycle which is shorter than C. Since C is a shortest cycle satisfying (*), one of the conditions of (***) must hold.

By the construction in (***), q_m subtends one fewer edges of C than q_{m-1}. Therefore q_p subtends $j-i+1-(p+1)=2$ edges of C which we label as e_n and e_{n+1}. (That is to say, $q_p=(R_{n-1},R_{n+1})$). As advertised, $<e_n,q_p>$ and $<e_{n+1},q_p>$. We must now show $<q_m,q_{m-1}>$ for $0<m\leq p$. We proceed by induction backwards from p.

There are two cases to consider in the basis.

Case1: $q_{p-1}=(R_{n-1},R_{n+2})$

Case2: $q_{p-1}=(R_{n-2},R_{n+1})$

(see figure A.2)

In case 1, $[q_{p-1},e_{n+2}]$ by the second condition of (***). If $<q_{p-1},e_{n+2}>$ then $<q_p,e_{n+2}>$ since $q_p,e_{n+2},q_{p-1}=(R_{n-1},R_{n+1}),(R_{n+1},R_{n+2}),(R_{n+2},R_{n-1})$ form a triangle. But then $<e_{n+1},e_{n+2}>$ by the transitivity of <,> violating (*). Therefore $<e_{n+2},q_{p-1}>$. But then $<q_p,q_{p-1}>$ by the triangle lemma, as required. The

reasoning in case 2 and the induction step is identical, subject to the required subscript changes.

We can now deduce $<e_{n+1}, q_0>$. But we also have

$$\begin{aligned} <e_i, e_j> &\rightarrow (R_{i-1} \cap R_i) \supseteq (R_{j-1} \cap R_j) \\ &\rightarrow (R_{i-1} \cap R_j) \supseteq (R_{i-1} \cap R_i \cap R_j) \supseteq (R_{j-1} \cap R_j) \\ &\rightarrow <q_0, e_j> \end{aligned}$$

Therefore $<e_{n+1}, e_j>$ which violates (**). ∎