OPTIMIZING CONJUNCTIVE QUERIES THAT CONTAIN UNTYPED VARIABLES

by

D. S. Johnson
and
A. Klug

Computer Sciences Technical Report #467

February 1982

Optimizing Conjunctive Queries that Contain Untyped Variables

D. S. Johnson

Bell Laboratories
University of Wisconsin

A. Klug

University of Wisconsin

ABSTRACT

This paper addresses questions of efficiency in relational databases. We present polynomial time algorithms for minimizing and testing equivalence of what we call "fan-out free" queries. The fan-out free queries form a more general and more powerful subclass of the conjunctive queries than those previously studied, as they can be used to express questions about transitive properties of databases, questions that are impossible to express if one operates under the assumption, implicit in previous work, that each variable has an assigned "type," and hence can only refer to one fixed attribute of a relation. Our algorithms are graph-theoretic in nature, and the equivalence algorithm can be viewed as solving a special case of the graph isomorphism problem (by reducing it to a series of labelled forest isomorphism questions).

Optimizing Conjunctive Queries that Contain Untyped Variables

D. S. Johnson

Bell Laboratories
University of Wisconsin

A. Klug

University of Wisconsin

1. INTRODUCTION

There has been much recent work addressing the problem of optimizing queries in the relational database model of Codd [5]. Such work has been motivated by the fact that, in the relational data model, it is easy to express queries whose natural implementation would be very inefficient. Thus there could be great value in a method for converting queries into equivalent ones with more efficient implementations. From a complexity-theoretic point of view, the seminal papers in this area are those of Chandra and Merlin [4] and Aho, Sagiv, and Ullman [2,3], which concentrated on the natural class of "conjunctive queries."

For conjunctive queries (which we shall define later), the main factor in the efficiency of the implementation is the number of conjuncts contained in the query. Thus the query optimization problem becomes simply the problem of finding an equivalent query involving the minimum possible number of conjuncts. Moreover, the problem of telling whether two minimal queries are equivalent can be reduced to a simple syntactic test. Unfortunately, as shown in [4], that "simple syntactic test" is equivalent to the presumably intractable problem of graph isomorphism, and the conjunct minimization problem is NP-hard, and hence even more likely to be intractable. (For a discussion of NP-hardness and intractability, see [1,6]).

Chandra and Merlin [4] faced this obstacle by designing exponential time algorithms for the problem, arguing quite reasonably that since queries might well be applied to very large databases, the savings resulting from a more efficient implementation of a query might well justify spending a large amount of time optimizing the query (that is, exponential time with respect to query length may still be smaller than, say, linear, quadratic, or some higher order polynomial time with respect to database size). Also, the same query may be run many times, thus amortizing the cost of optimization.

Aho, Sagiv, and Ullman [2,3] took the alternate route of looking for special cases that could be solved efficiently, i.e., in polynomial time. As a first step, they restricted attention to *typed* queries, i.e., queries in which each variable has a specific attribute of the database associated with it as its "type," and can only refer to entries in columns of the database labelled by that attribute. This is a significant restriction, since it seriously interferes with our ability to derive transitive information from databases, for instance, to ask for (child, grandparent) pairs given a relation containing only (child, parent) pairs. Given this basic restriction and an additional technical restriction to only those queries representable by "simple tableaux," they derived a class of queries for which minimization and equivalence testing can both be performed in polynomial time: minimization in time $O(n^3)$ [3] and equivalence testing in time $O(n^4)$ [2]. Subsequent work by Sagiv [7] has reduced both running times to $O(n^2)$. Sagiv and

A preliminary version of this paper appeared, under a slightly different title, in the *Proceedings of the 22nd Annual Symposium on Foundations of Computing*, IEEE Computer Society, Los Angeles, Calif., 1981.

Yannakakis [8] have also introduced two new classes of typed queries for which the problems can be solved in polynomial time, again $O(n^2)$ for minimization [7] and $O(n^2)$ for equivalence [8].

In this paper we show that polynomial time algorithms exist even when the restriction to typed queries is dropped, although new techniques are required for these more general algorithms. Our running times are, coincidentally, the same as those in the original Aho, Sagiv, and Ullman papers [2,3].

In Section 2 we review the basic definitions and illustrate the limitations of typed queries, as well as the other restrictions of [2,3,7,8]. In Section 3 we define a class of conjunctive queries, the *fan-out free* queries, which allow *un*typed variables, and can be viewed as a generalization of the other classes, even in the typed case. The "fan-out free" property is specified in terms of an *implication graph* that is defined for any conjunctive query. In Section 4 we show how the minimization problem for fan-out free queries can be transformed into a series of connectivity tests on this implication graph and thus solved in time $O(n^3)$.

The algorithm of Section 4 is in a sense just a generalization and extension of the ideas behind the earlier techniques of [3,7] to a much wider class (about the widest possible such class that has a simple structural characterization). The techniques used for equivalence testing in [2,8] do not, however, extend in any nice way once untyped variables are allowed. We obtain an algorithm by showing that the problem, although equivalent to graph isomorphism if we consider general (minimized) conjunctive queries, can be reduced to the well-solved problem of labeled tree isomorphism in the case of fan-out free queries. The algorithm runs in time $O(n^4)$ and is presented in Section 5. Most of the work in the section is involved in showing that an appropriately labeled breadth-first spanning forest for the implication graph of such a query can be viewed as a canonical form for the query.

The paper concludes in Section 6 with an examination of possible extensions and improvements on our algorithms, along with other directions for future research.

2. A FEW DEFINITIONS

A *relation* R can be viewed as a finite two-dimensional table with columns labeled by distinct *attributes*. Each attribute A_i has a *domain* $D(A_i)$, and entries in a column labeled by by A_i must be elements of the domain $D(A_i)$. The *relation scheme* for a relation R is the sequence of attributes labelling its columns, and may be viewed as an ordered subset of the set of all attributes. Since the order of the *rows* in a relation has no significance, we can view a relation with scheme $(A_1,A_2,...,A_k)$ as a finite subset of $D(A_1) \times D(A_2) \times \cdots \times D(A_k)$. A *database* is a finite sequence of tables. The *relation schema* for a database is the sequence of relation schemes for the tables it contains. In what follows we shall assume for simplicity of notation that all databases under consideration have the same relation schema $(S_1,...,S_s)$.

We shall also assume that there are no data dependencies. "Functional dependencies" can be handled just as in [2,3], by means of a preprocessing step prior to the application of our algorithms. More complicated dependencies, such as multi-valued dependencies or the even more general "algebraic" dependencies of [9], give rise to much added complexity, and we follow [2,3] in leaving these complexities for future work.

A *query* can be viewed as a function from databases to relations. A query Q has *target scheme* S_Q if it maps databases to relations with relation scheme S_Q. A *conjunctive query* Q can be specified by $S_Q = (A_1, \ldots, A_p)$ and the following: (1) A set $X_Q = \{x_1,...,x_p\}$ of *distinguished variables* (DV's), (2) a set $Y_Q = \{y_1,...,y_q\}$ of *non-distinguished variables* (NDV's), and (3) a set $C_Q = \{c_1, \ldots, c_r\}$ of distinct *conjuncts*, each conjunct c_i being associated with a relation $R(c_i)$ of the underlying relation schema and having the form $(c_i[1],...,c_i[m])$, where $m = Length(c)$ is the number of columns (attributes) in $R(c_i)$ and each $c_i[j]$ is either a DV, an NDV, or a *constant*, i.e., an element of the domain of the j^{th} attribute of $R(c_i)$.

Given a database B and a conjunctive query Q, the relation constructed when Q is applied to B is

$Q(B) = \{(x_1,...,x_p) : x_i \in D(A_i)$ and there exist $y_1,...,y_q$ such that for all $c_i \in C_Q$, $(c_i[1],...,c_i[m])$, with the valuations of the DV's and NDV's substituted in, is a row in relation $R(c_i)$ of $B\}$

For simplicity, we shall often specify conjunctive queries implicitly, using a format similar to the above, using "$R(x,y)$" to stand for the statement that (x,y) is a row in relation R. For example:

$Q = \{(x_1,x_2,x_3)$: there exist y_1, y_2 such that $R_1(x_1,y_1,x_2)$ and $R_2(x_1,y_2,3)\}$.

The identities of the DV's, NDV's, constants, and conjuncts are all easily derived from this shorthand.

Chandra and Merlin show in [4] how a query Q can be computed using the operations of "selection," "restriction," and "generalized join," applied to the database which is the query's argument. The major factor in determining the running time for the resulting algorithm is the number of generalized joins performed, and this number is one less than the number of conjuncts in C_Q. Thus the query minimization problem for conjunctive queries is simply the problem of finding a query equivalent to Q which has the least possible number of conjuncts.

Until now, the classes of queries for which efficient minimization algorithms have been derived have all only allowed *typed* variables. A DV or NDV u is typed if it is associated with a unique attribute *type*(u), i.e., if $c_i[h] = u$, then the h^{th} attribute of relation $R(c_i)$ is *type*(u).

If a query contains only typed variables, then the underlying relation schema might as well be assumed to have disjoint attribute domains, i.e., $D(A_i) \cap D(A_j) = \varnothing$ for all pairs $A_i \neq A_j$ of attributes in the schema. Even though in practice many databases have overlapping domains, typed queries cannot make use of this information (except insofar as constants in the query can be associated with more than one attribute). Consider the relation R_1 illustrated in Figure 1, with its first column labelled by the attribute "parent" its second labelled by the attribute "child", and $R_1(x,y)$ if and only if x is the parent of y. The domain for each attribute is the set of people, and of course a person can be both a parent and a child. This suggests the possibility of deriving from R_1 the grandparent relation R_G, where R_G has grandparent and child columns, and $(x,y) \in R_G$ if x is y's grandparent. This can be done quite simply using untyped variables:

$\{(x_1,x_2)$: there is a y_1 such that $R_1(x_1,y_1)$ and $R_1(y_1,x_2)\}$

However, no typed query will suffice, because such a query cannot use the fact that a parent can be a child.

One cannot get around this problem by simply giving both columns of the relation the same name, say "person," because our basic definitions say that all columns of a relation must be named by distinct attributes, and this fact is crucial to the algorithms of [2,3,7,8]. A second approach, that of using auxiliary tables, *does* allow us to construct the grandparent relation with a typed query, but the results are awkward and inefficient: In the current example what we need is a second relation R_2, which captures the "equivalence" of parents and children in a typed fashion. R_2 has a parent column and a child column, and $(x,y) \in R_2$ if and only if x and y refer to the same person. See Figure 2. We then can specify the grandparent relation R_G with the following typed query:

$\{(x_1,x_2)$: There are y_1 and y_2 such that $R_1(x_1,y_1)$, $R_2(y_2,y_1)$, and $R_1(y_2,x_2)\}$

Note that the use of this subterfuge not only makes the query harder to understand, but also introduces a new conjunct, and hence a new "join" to the query. We thus can circumvent the restriction to typed queries, but only at a cost of lowered query efficiency.

Another problem is that the three subclasses of typed queries for which algorithms have

been derived each impose an additional constraint on the queries they contain, and these constraints interfere with the use of equivalence tables. The first two of these restrictions involve the concept of a *repeated variable*, which is defined to be a NDV that occurs more than once in the conjuncts of Q.

(1) If an attribute has a repeated variable associated with it, then there is no other repeated symbol (variable or constant) associated with it.

(2) No conjunct contains more than one occurrence of a repeated variable.

The first of these restrictions gives rise to the class of queries with "simple tableaux," as defined and discussed in [2,3]. The second gives rise to the first of the two classes introduced in [7,8].

The reader may verify that (2) makes it impossible to construct the grandparent relation using a typed query, even if equivalence tables are allowed (and rules out the "great-grandparent" relation R_{GG} even if untyped variables are allowed). Restriction (1) allows grandparents, but rules out great-grandparents using either equivalence tables or untyped variables. The third restriction, corresponding to the second class from [7,8] and using the notion of "covering" which will be defined shortly, also prevents the construction of great-grandparents:

(3) No conjunct of Q is "covered" by more than one conjunct in C_Q other than itself.

Although the class of "fan-out free" conjunctive queries for which our algorithms are designed also obeys a restriction (a weaker one), this new class contains the queries required for constructing *any* fixed-height ancestor relation, and this should be indicative of its increased power over the previous classes.

A final objection to the restriction to typed queries is more theoretical in nature: the natural symmetry between conjunctive queries and relational algebra is lost. As shown by Chandra and Merlin [4], the set of relations constructible by conjunctive queries is precisely the set of relations that can be constructed using the operations of select, project, and generalized join. No analogous characterization has been found for typed conjunctive queries, unless one allows an unnatural extension of the concept of "relation" [9].

Thus it is with no great sense of loss that we abandon the restriction to typed queries. Since we are dropping the assumption, there is no real restriction in assuming for simplicity that all attributes have the same domain D, and so we shall follow [4] in doing so. The next section presents the series of definitions which leads to the definition of "fan-out free."

3. IMPLICATION GRAPHS AND FAN-OUT FREE QUERIES

Suppose two queries Q and Q' have the same target scheme, and hence the same distinguished variables. Let U_Q be the union of X_Q, Y_Q, and D, and define $U_{Q'}$ similarly. A *symbol mapping* from Q to Q' is a function $f:U_Q \rightarrow U_{Q'}$ which leaves all DV's and constants fixed. If c and c' are conjuncts in Q and Q' respectively, we say that c' *covers* c if $R(c) = R(c')$ and there is a symbol mapping from Q to Q' that sends c to c'. A *conjunct mapping* from Q to Q' is a function $g:C_Q \rightarrow C_{Q'}$ such that for all $c \in C_Q$, $g(c)$ covers c.

A fundamental concept, both for query minimization and equivalence testing, is that of a *homomorphism*. A *homomorphism* from Q to Q' is a symbol mapping $f:U_Q \rightarrow U_{Q'}$ that induces a well-defined conjunct mapping. Alternatively, we may view a homomorphism as a conjunct mapping $g:C_Q \rightarrow C_{Q'}$ that induces a well-defined symbol mapping. In what follows we shall often view a homomorphism as *both* of the above, i.e., a function $h:Q \rightarrow Q'$ defined on both U_Q and C_Q, whose symbol mapping half induces its conjunct mapping half, and vice versa.

The importance of homomorphisms comes from the following result.

Theorem 3.1 [4]. Two queries Q and Q' are equivalent (as functions defined on databases) if and only if there are homomorphisms $h:Q \rightarrow Q'$ and $h':Q' \rightarrow Q$.

Thus in both minimization and equivalence testing, our main task will be the search for

homomorphisms. In fact, it will turn out that most of our work is involved with finding *self-homomorphisms*, i.e., homomorphisms from a query Q to itself. In order to model potential self-homomorphisms, we introduce the notion of an *implication graph*. Given any conjunctive query Q, the *implication graph* of Q is a bipartite graph $G[Q] = (V[Q],E[Q])$ defined as follows:

The vertices are of two types. First, there is a set $V_C[Q]$ of *conjunct-pair* vertices, containing a vertex $<c,\theta>$ for each conjunct c in C_Q and a vertex $<c,c'>$ for each pair of (not necessarily distinct) conjuncts in C_Q such that c' covers c. (θ is a special symbol indicating that c is *not* covered by any relevant conjunct). Second, there is a set $V_S[Q]$ of *symbol-pair* vertices, containing a vertex $<y,z>$ for each NDV $y \in Y_Q$ and each $z \in U_Q$, such that for some conjunct-pair vertex $<c_1,c_2>$ and index j, $1 \le j \le Length(c_1)$, $c_1[j] = y$ and $c_2[j] = z$. (In the future we shall often omit the "$[Q]$" from $V_C[Q]$ and $V_S[Q]$, so long as there is no chance of confusion). Each edge in $E[Q]$ joins a symbol-pair vertex to a conjunct-pair vertex. There is an edge between $<y,z>$ and $<c,\theta>$ if and only if there is a j, $1 \le j \le Length(c)$, such that $c[j] = y$ and for all c' such that c' covers c, $c'[j] \neq z$. There is an edge between $<y,z>$ and $<c,c'>$ if and only if there is a j, $1 \le j \le Length(c)$, such that $c[j] = y$ and $c'[j] = z$. Figure 3 illustrates the implication graph for the conjunctive query that constructs the great-great-grandparent relation from the parent-child relation R_1.

Using the implication graph $G[Q]$, we can test whether a symbol mapping (or a conjunct mapping) of Q to itself corresponds to a self-homomorphism. A subset V' of the symbol-pair vertices of $G[Q]$ is said to satisfy the *symbol mapping property* (*partial symbol mapping property*) if each $y \in Y_Q$ occurs as the first component of exactly one (at most one) of its vertices. It is easy to see that any symbol mapping capable of yielding a self-homomorphism must correspond to a set V' satisfying the symbol mapping property: the subset corresponding to the symbol mapping f is simply $V_{S,f} = \{<y,z> : y \in Y_Q \text{ and } f(y)=z\}$.

The *conjunct mapping property* (*partial conjunct mapping property*) is defined analogously for subsets of the conjunct-pair vertices of $G[Q]$, none of whose second components is θ. The subset corresponding to the conjunct mapping g is simply $V_{C,g} = \{<c,c'> : c \in C_Q \text{ and } g(c)=c'\}$. A self-homomorphism h, viewed as a joint symbol and conjunct mapping, will then correspond to the set $V_{S,h} \cup V_{C,h}$ of symbol-pair and conjunct-pair vertices, where each half induces the other and obeys the appropriate mapping property.

A set V' of symbol-pair vertices induces a set V'' of conjunct-pair vertices if it contains all the neighbors of each vertex in V''. We capture the notion of the set of symbol-pair vertices induced by a set V' of conjunct-pair vertices by using the concept of *implication* in $G[Q]$:

A conjunct-pair vertex *implies* all of its neighbors, while symbol-pair vertices also imply neighbors, but are more selective: A symbol-pair vertex of the form $<y,y>$ implies nothing. A symbol-pair vertex which is adjacent to a conjunct-pair vertex of the form $<c,\theta>$ implies all vertices of this form to which it is adjacent, and nothing else. A symbol-pair vertex of the form $<y,z>$, $y \neq z$, implies a neighboring conjunct-pair vertex $<c,c'>$ if and only if this is the only conjunct-pair vertex having c as first component to which $<y,z>$ is adjacent.

The following observation is immediate (and only uses about half of the above definition).

Lemma 3.1. A conjunct mapping g of Q to itself is a self-homomorphism of Q if and only if the set of symbol-pair vertices inplied by $V_{C,g}$ obeys the symbol mapping property.

The definition of implication is more complicated than is required by Lemma 3.1 because we wish to use the implication graph to determine whether certain *partial* conjunct mappings can be extended to self-homomorphisms. To this end we introduce the notion of *implication closure*. If V' is a subset of the vertices of $G[Q]$, then the implication closure $\overline{V'}$ of V' is the smallest superset of V' that is closed under implication, i.e., that contains all the vertices implied by its vertices. The implication closure of $V_{C,g}$, where g is a partial conjunct mapping of Q to itself, is designed to include all the forced consequences of g.

Unfortunately, in the general case for conjunctive queries, these "forced consequences"

do not provide enough information to tell us whether g can be extended to a self-homomorphism. There is a "fan-out" of possibilities, and one would have to examine them all in order to verify extendability. Thus in what follows we impose a restriction on queries that guarantees that such fan-out cannot occur. We say that a symbol-pair vertex $\langle y,z\rangle$ is *fan-out free* if either (a) all its neighbors have the *same* first component or (b) all its neighbors have *distinct* first components. In the latter case $\langle y,z\rangle$ implies all of its neighbors, in the former it implies none of them (except in the special case where it has only one neighbor). A conjunctive query Q is *fan-out free* if and only if every symbol-pair vertex $\langle y,z\rangle$, $y \neq z$, in V_S is either adjacent to a vertex of the form $\langle c,\theta\rangle$ or is fan-out free.

To see that this restriction prevents the fan-out of possibilities, consider the consequences of adding a conjunct-pair $\langle c,c'\rangle$ to a partial conjunct mapping for a fan-out free query. If $\langle c,c'\rangle$ implies $\langle y,z\rangle$ and $y \neq z$ then either (a) $\langle y,z\rangle$ is adjacent to a vertex of the form $\langle c_i,\theta\rangle$ and hence no extension to a full self-homomorphism is possible, (b) $\langle y,z\rangle$ is adjacent only to other conjunct-pairs with c as first component, none of which can be part of the same homomorphism as $\langle c,c'\rangle$, or (c) *all* the other conjunct-pairs adjacent to $\langle y,z\rangle$ are implied by $\langle y,z\rangle$ and must be present in any self-homomorphism containing $\langle c,c'\rangle$. If a symbol-pair vertex $\langle y,z\rangle$ with fan-out were allowed, it might have neighbors $\langle c_1,c_2\rangle$ and $\langle c_1,c_3\rangle$, $c_2 \neq c_3$, and we would have a *choice* as to which one to add in attempting to extend our partial conjunct mapping.

Note that we do allow fan-out at vertices of the form $\langle y,y\rangle$, but under our definitions these vertices do not imply any of their neighbors, and the following theorem will show that they do not need to. Let us say that a subset V' of the conjunct-pair vertices of an implication graph $G[Q]$ is *transitively closed* if the implication closure of V' contains no new conjunct-pair vertex, i.e., if $\overline{V'} \cap V_C = V'$. A transitively closed set V' of conjunct-pair vertices obeys the *partial homomorphism property* if it satisfies the following two conditions:

(C1) There is no conjunct-pair vertex with θ as second component in V'.

(C2) The set V'' of symbol-pair vertices in $\overline{V'}$ obeys the partial symbol mapping property.

Observe that (C2) implies that V' itself obeys the partial conjunct mapping property, since if V' contained two distinct conjunct-pairs with the same first component, then V'' would have to contain two distinct symbol-pairs with the same first component. Theorem 3.2 validates our claim that, for fan-out free queries, we can tell whether a partial conjunct mapping extends to a full self-homomorphism merely by computing its implication closure:

Theorem 3.2. Let Q be a fan-out free query and suppose V' is a transitively closed subset of the conjunct-pair vertices of $G[Q]$ that obeys the partial homomorphism property. Then the set V^* of vertices obtained by adding to $\overline{V'}$ all symbol-pair vertices $\langle y,y\rangle$, where y does not occur as a first component in $\overline{V'}$, and all conjunct-pair vertices $\langle c,c\rangle$, where c does not appear as a first component in $\overline{V'}$, corresponds to a self-homomorphism of Q.

Proof. Let f and g be the symbol mapping and conjunct mapping corresponding to $V_S{}^* = V^* \cap V_S$ and $V_C{}^* = V^* \cap V_C$ respectively. By (C2) f obeys the symbol mapping property and, by (C1) and our observation following the definition of the partial homomorphism property, g obeys the conjunct mapping property. Thus by Lemma 3.1 all we need show is that $V_C{}^*$ implies $V_S{}^*$. This can fail to happen only if (i) $V_C{}^*$ implies some $\langle y,z\rangle$ not in $V_S{}^*$ or (ii) $V_C{}^*$ fails to imply some $\langle y,z\rangle$ which *is* in $V_S{}^*$. We show that both possibilities lead to contradictions.

(i). Suppose $V_C{}^*$ implies a symbol-pair $\langle y,z\rangle$ which is not in $V_S{}^*$. Then $\langle y,z\rangle$ must be implied by some conjunct-pair $\langle c,c'\rangle$ that is in V^* but not in V', and hence has $c' = c$. This means that $z = y$. The fact that $\langle y,y\rangle$ is not in $V_S{}^*$ means that there is some $\langle y,z'\rangle$, $z' \neq y$, which is in $V_S{}^*$. The fact that c contains an occurence of y means that there is some conjunct-pair vertex $\langle c,c''\rangle$, $c'' \neq c$, adjacent to $\langle y,z'\rangle$ in G_Q. The fact that $\langle c,c\rangle$ is in V^* means that $\langle y,z'\rangle$ does not imply $\langle c,c''\rangle$. Since $\langle y,z'\rangle$ is not adjacent to any conjunct-pair vertex with θ as second component by (C1), and since $z' \neq y$, the fact that Q is fan-out free thus allows us to conclude that all vertices adjacent to $\langle y,z'\rangle$ in G_Q have c as

first component. In particular, the conjunct-pair in V' that implies $<y,z'>$ has c as first component, in contradiction to the fact that $<c,c>$ is in V^*.

(ii). Suppose V_C^* fails to imply some symbol-pair $<y,z>$ in V_S^*. Then we must have that $<y,z>$ is not in V' and hence $z = y$. By our definition of the implication closure, there must be some conjunct c that contains an occurrence of y. We cannot have $<c,c>$ in V_C^*, or else V_C^* would imply $<y,y>$. Hence there is some $<c,c'>$ in V_C^*, $c' \neq c$. The vertex $<c,c'>$ cannot be adjacent to $<y,y>$ in G_Q (or else it would imply it), but it must be adjacent to *some* symbol-pair $<y,z'>$ with y as first component. But this means that $<y,z'>$ is in V_S^* and $z' \neq y$, and hence we cannot have added $<y,y>$ to V', again a contradiction.

Thus both cases are impossible and the theorem is proved. □

We thus see that our restriction to fan-out free queries does what we want it to. The fact that it is not too severe a restriction can be gleaned from the next two theorems.

Theorem 3.3. Any typed query Q satisfying restriction (1) or (3) of Section 2 is fan-out free, and there exist fan-out free typed queries that obey neither (1) nor (3).

Proof. In order for Q to fail to be fan-out free, there must be a symbol-pair vertex $<y,z>$ in $G[Q]$ such that $y \neq z$ and $<y,z>$ is adjacent to three conjunct-pair vertices $<c_1,c_2>$, $<c_3,c_4>$, and $<c_3,c_5>$, where $c_1 \neq c_3$ and $c_4 \neq c_5$. Since $y \neq z$, this would mean that c_3 was covered by two conjuncts other than itself, and so Q would violate (3). It would also mean that y was a repeated variable (occurring in both c_1 and c_3) and z was a second repeated symbol (occurring in c_4 and c_5) associated with the same attribute, thus violating (1). Hence any query satisfying (1) or (3) must be fan-out free.

An example of a typed query which is fan-out free but violates both (1) and (3) is the query which constructs the great-grandparent relation using the parent-child relation R_1 and the equivalence relation R_2 from Section 2:

$$\{(x_1,x_2) : \text{there exist } y_1, y_2, y_3, \text{ and } y_4 \text{ such that } R_1(x_1,y_1), R_2(y_2,y_1), R_1(y_2,y_3), R_2(y_4,y_3), \text{ and } R_1(y_4,x_2)\}$$

The third conjunct of this query is covered by both the first and the fifth, thus violating (3), and the "parent" attribute has two repeated variables (y_1 and y_3), thus violating (1). To see that the query is fan-out free, note that the only conjunct-pair vertices $<c,c'>$ in $G[Q]$ with $c \neq c'$ are $<c_2,c_4>$, $<c_4,c_2>$, $<c_3,c_1>$, and $<c_3,c_5>$, and that no $<y,z>$, $y \neq z$, is adjacent to more than two of them or to *any* conjunct-pair of the form $<c,c>$ (at least three neighbors are required for a symbol-pair vertex to have "fan-out"). □

Thus, the class of fan-out free queries is more general than either of the classes defined by the restrictions (1) and (3), even when restricted (as *they* are) to typed variables. The relationship between fan-out free queries and those obeying restriction (2) of Section 2 is less straightforward. Although (2), by forbidding more than one repeated variable in any conjunct, rules out any sophisticated use of transitivity, it does allow some queries which are not fan-out free. A simple example would be

$$\{(x_1) : \text{there exist } y_1, y_2, y_3, \text{ and } y_4 \text{ such that } R_1(y_1,x_1), R_1(y_1,y_2), R_1(y_3,x_1), \text{ and } R_1(y_3,y_4)\}$$

The query obeys (2) since the only repeated (non-distinguished) variables are y_1 and y_3. It is not fan-out free since $<y_3,y_1>$ is adjacent to $<c_3,c_1>$, $<c_4,c_1>$, and $<c_4,c_2>$.

However, note that the above query is equivalent to the fan-out free query containing the single conjunct $<y_1,x_1>$. In fact, it is not difficult to prove the following theorem (using Lemma 4.2 of the next section).

Theorem 3.4. Any minimal query satisfying restriction (2) of Section 2 is fan-out free.

Thus any question that can be phrased using a conjunctive query obeying restrictions (1), (2), or (3) can be phrased using a fan-out free query, and fan-out free queries can express

many questions that would be impossible to ask if any one of those restrictions must be obeyed.

4. MINIMIZING FAN-OUT FREE QUERIES

Our minimization algorithm, like those of [3,7], is based on the following result from [4]:

Lemma 4.1. If a conjunctive query Q is equivalent to a conjunctive query Q' that has fewer conjuncts, then there is a self-homomorphism of Q that sends the conjuncts of Q to a proper subset of themselves (a *shrinking self-homomorphism* of Q).

Proof. By Theorem 3.1, if Q is equivalent to Q' then there exist homomorphisms $h:Q\rightarrow Q'$ and $h':Q'\rightarrow Q$. The desired self-homomorphism is h composed with h'. □

We also use the following easy observation:

Lemma 4.2. If h is a self-homomorphism of Q then the set of conjuncts in the image of h represents a query Q' which is equivalent to Q.

Proof. By Theorem 3.1 we need to show that there are homomorphisms in both directions. We already have the homomorphism h which goes from Q to Q'. The homomorphism in the other direction is simply the identity self-homomorphism on Q, restricted to the conjuncts and NDV's in Q'. □

Our minimization algorithm will be based on a subroutine S which, given Q, determines whether it has a shrinking self-homomorphism, and, if so, returns one. We use S in the following loop.

1. Set $Q^* = Q$.
2. While S applied to Q^* yields a shrinking self-homomorphism h, set $Q^* = h(Q^*)$.
3. Return Q^* (If S applied to Q^* does not yield a shrinking self-homomorphism, then Q^* is our desired minimum equivalent query).

Note that if Q has N conjuncts, subroutine S will be invoked at most N times, since each successful application reduces the number of conjuncts, and we always must have at least one conjunct left.

We thus have reduced to problem of conjunctive query minimization to that of finding shrinking self-homomorphisms. In the case of fan-out free queries, we can use the results of the last section in doing the "finding":

Lemma 4.3. Suppose Q is a fan-out free query. Then the following two statements are equivalent.

(a) Q has an shrinking self-homomorphism.

(b) There is a conjunct-pair vertex $<c,c'>$ in $G[Q]$ with $c\neq c'$ and $c'\neq\theta$, such that the set V' of conjunct-pair vertices in the implication closure of $\{<c,c'>\}$ satisfies the partial homomorphism property and the set $C_1(V')$ of first components of conjunct-pairs in V' does not equal the set $C_2(V')$ of second components.

Proof. That (b) implies (a) follows immediately from Theorem 3.3. By that theorem, V' extends to a self-homomorphism h of Q which is the identity on all conjuncts not in $C_1(V')$. Since there is a conjunct in $C_1(V') - C_2(V')$, that conjunct will not be in the image of C_Q under h, and so h is an shrinking self-homomorphism.

To prove the implication in the other direction, let us suppose that h is a shrinking self-homomorphism of Q, and let $V' = V_{C,h}$ be the associated set of conjunct-pair vertices in $G[Q]$. Consider the implication closure $\overline{V'}$, which is simply $V'\cup V_{S,h}$ since h is a self-homomorphism. By the definition of implication closure, $\overline{V'}$ is the union of the implication closure of $\{<c,c'>\}$, for all $<c,c'>$ in V'. Let $\overline{V}[c,c']$ be the set of conjunct-pairs in the implication closure of $<c,c'>$. If $C_1(\overline{V}[c,c']) = C_2(\overline{V}[c,c'])$ for all $<c,c'>$ in V', then we must have $C_1(V') = C_2(V')$ and so h would not be a shrinking self-homomorphism. Thus there is some $<c,c'>$ such that $\overline{V}[c,c']$ has a set of first components *not* equalling its set of

second components. We claim that this is the desired conjunct-pair. Clearly $c \neq c'$, since the implication closure of $<c,c>$ is simply that vertex by itself, and $C_1(\overline{V}[c,c]) = C_2(\overline{V}[c,c]) = \{c\}$. Furthermore, $c' \neq \theta$ since V' contains no vertex with θ as second component. The set $\overline{V}[c,c']$ does not contain any conjunct-pairs with θ as second component for the same reason, so it satisfies (C1) of the partial homomorphism property. Finally, $\overline{V}[c,c']$ satisfies (C2) since it is contained in $\overline{V}'$. Thus $\overline{V}[c,c']$, being transitively closed, satisfies the partial homomorphism property, and $<c,c'>$ is the desired conjunct-pair. □

Lemma 4.3 is enough to give us a polynomial time algorithm for minimizing fan-out free queries. We can implement subroutine S by (1) constructing the implication graph for Q, (2) finding the implication closure of each of its conjunct-pair vertices $<c,c'>$, $c \neq c'$, (3) testing each implication closure to see whether its conjunct-pairs satisfy property (b) of the lemma, and finally (4) using Theorem 3.2 to generate the corresponding shrinking self-homomorphism when it exists.

As schematized above, our algorithm has a major potential inefficiency. If there are N conjuncts (and hence potentially $N(N-1)/2$ conjunct-pairs $<c,c'>$, $c \neq c'$), the time to find all the implication closures might be no better than $O(N^2)$ times the size of $G[Q]$. Such a bound could be attained if many of the implication closures were large and overlapped, causing us to examine the same part of the graph over and over again. In order to avoid this, we make use of one more lemma about implication graphs of fan-out free queries.

Lemma 4.4. If Q if a fan-out free query and $<c_1,c_2>$, $<y,z>$, and $<c_3,c_4>$ are vertices of $G[Q]$ such that $<c_1,c_2>$ implies $<y,z>$ and $<y,z>$ implies $<c_3,c_4>$, then $<c_3,c_4>$ implies $<y,z>$ and $<y,z>$ implies $<c_1,c_2>$.

Proof. For the assumed implications to hold, we must have that $<y,z>$ is adjacent to both $<c_1,c_2>$ and $<c_3,c_4>$. Thus $<c_3,c_4>$ implies $<y,z>$, since a conjunct-pair implies all symbol-pairs to which it is adjacent. The argument for the second implication goes as follows. Since $<y,z>$ implies $<c_3,c_4>$, we know that $<y,z>$ is not adjacent to any conjunct-pairs with θ as second component, $y \neq z$, and $c_3 \neq c_1$. Since Q is fan-out free, this means that *all* conjunct-pairs adjacent to $<y,z>$ have distinct first components, and thus $<y,z>$ implies $<c_1,c_2>$. □

As a consequence of this lemma, we note that if $<c_1,c_2>$ is in the implication closure of $\{<c_3,c_4>\}$ and $c_2 \neq \theta$, then $<c_3,c_4>$ is in the implication closure of $\{<c_1,c_2>\}$, and, in fact, the two implication closures are equal. Thus, in the process of generating the implication closures of individual $<c,c'>$'s, we never need generate the implication closure of a conjunct-pair $<c,c'>$ which has already appeared in the implication closure of some earlier conjunct-pair. Hence we only need look at each conjunct-pair vertex once in the entire process. A symbol-pair vertex *can* appear in more than one distinct implication closure, if it is one of those symbol-pairs that, for some reason, does not imply all its neighbors. However, it will only be encountered a number of times bounded by the number of its neighbors, since each neighbor is itself in at most one implication closure. Thus it is easy to see that the total amount of work required to generate the relevant implication closures under this modified scheme is proportional to the size of the implication graph, not N^2 times that size.

For those uninterested in writing special programs to generate implication closures, we note that we can modify the implication graph so that the relevant implication closures can be found using a standard algorithm for generating the connected components of a graph. Two basic operations need to be performed:

(O1) Coalesce into a single vertex $<\theta>$ all symbol-pair vertices which are adjacent to any conjunct-pair with θ as second component, and delete all such conjunct-pair vertices. The new vertex $<\theta>$ is adjacent to all remaining conjunct-pairs to which the $<c,\theta>$ vertices were originally adjacent.

(O2) For each remaining symbol-pair vertex $<y,z>$ where either $y = z$ or all the neighbors of $<y,z>$ have the same first component, split the vertex $<y,z>$ into a number of copies of itself equal to the number of conjunct-pair vertices adjacent to it, and let each of these

copies be adjacent to a corresponding one of those conjunct-pairs.

We leave to the reader the straightforward task of verifying that the connected components of the modified graph correspond to the implication closures of the sets $\{<c,c'>\}$ in $G[Q]$, with the union of all implication closures that contain a $<c,\theta>$ vertex corresponding to the single connected component containing $<\theta>$. Since the connected components of a graph can be generated in time which is linear in the size of the graph [1], and since the modified graph can be constructed in time proportional to the size of $G[Q]$, this graph-connectivity approach will be about equivalent to the one already proposed.

Having found the relevant implication closures, we must then examine each to see whether its conjunct-pair vertices satisfy the partial homomorphism property and have a set of second components different from their set of first components. This can be done for each $V'=\overline{V}[c,c']$ in time proportional to the size of V', and hence the overall time for this part of the algorithm is also bounded by the size of $G[Q]$. We test V' as follows. Condition (C1) of the partial homomorphism property, that V' contain no $<c,\theta>$ vertex, can easily be checked in the claimed time. Condition (C2), that no two symbol-pair vertices of V' share the same first component, can be verified in time proportional to the number of such vertices by indexing into an array with entries for each NDV (we avoid initializing the table by using the trick described in Exercise 2.12 of [1]). A similar trick is used to verify that $C_1(V') \neq C_2(V')$.

To complete the analysis of the running time of our minimization algorithm, we must consider how large the implication graph $G[Q]$ can be, and how much time is required to generate it. As argued above, $G[Q]$ contains at most $O(N^2)$ conjunct-pair vertices., with at most N having any given conjunct c as first component. Each conjunct-pair vertex $<c,c'>$ is adjacent to at most $Length(c)$ symbol-pair vertices, one for each column of c. Thus the total number of symbol-pair vertices, and the total number of edges in $E[Q]$ are both at most N times the sum of the lengths of the conjuncts in C_Q, where this sum can be taken as proportional to n, the size of the query itself. Since $N \leq n$, we thus obtain $O(n^2)$ as a bound on the size of $G[Q]$. The time to generate $G[Q]$ is proportional to its size, since we can tell whether c' covers c in time proportional to to Length(c), and if so, generate all the edges incident on $<c,c'>$ at the same time, merely by comparing $c[i]$ and $c'[i]$, $1 \leq i \leq Length(i)$ (using our indexing trick to make sure that no earlier j had $c[j]=c[i]$ and $c'[j] \neq c'[i]$).

Thus the time for our subroutine **S** is $O(n^2)$ and hence the time for the overall minimization algorithm is $O(n^3)$. Note however that for queries that are already near-minimal, as we would expect them to be in practice, the time bound may reduce to $O(n^2)$, since the number of times **S** is applied is at most one more than the number of redundant conjuncts in Q.

In cases where the Q is far from minimal, we might hope to speed up the algorithm by a more sophisticated generation by **S** of the shrinking self-homomorphism h it returns. Clearly our goal should be to find a shrinking self-homomorphism which has the smallest image. One thing we could do is generate h from the $V'=\overline{V}[c,c']$ that has the largest value of $|C_1(V')-C_2(V')|$. This could be done without substantially increasing the overhead in **S**.

A more expensive approach would use the fact that some *set* of implication closures must correspond to a maximal shrinking self-homomorphism, i.e., one which sends C_Q to a minimum equivalent subset of itself. Thus we might attempt to combine implication closure's and see whether the combined implication closure (the union of two implication closures is itself an implication closure) satisfies the conditions of Lemma 4.3 and has a larger value of $|C_1(V')-C_2(V')|$ than either of its progenitors. The complexity involved in doing these tests can be reduced somewhat by using the fact that there is a maximal shrinking self-homomorphism which is the identity self-homomorphism when restricted to its image (this fact can be proved by considering high-order compositions of a maximal shrinking self-homomorphism h^* with itself). Even so, the cost of finding a maximal shrinking self-homomorphism in this manner would probably be prohibitive if there were a large number of distinct implication closures (it would probably greatly exceed the potential savings due to reducing the number of applications of **S** - we are flirting with NP-completeness here). The best hope is to direct the merging of implication closure's by heuristics, in hopes of finding a

near-maximal shrinking self-homomorphism at reasonable cost. We have not investigated this avenue fully, although it is not difficult to show that a greedy approach can leave one arbitrarily far from maximal.

5. EQUIVALENCE TESTING FOR FAN-OUT FREE QUERIES

The first step of our algorithm for testing whether two fan-out free queries Q and Q' are equivalent is to perform the minimization algorithm of the previous section on both of them. This reduces the equivalence problem to one of isomorphism, as can be seen from the following easily-proved lemma:

Lemma 5.1. Suppose Q and Q' are minimal queries. Then Q is equivalent to Q' if and only if there is a homomorphism from Q to Q' which is one-one and onto for both NDV's and conjuncts.

Note that by setting $Q = Q'$ in Lemma 5.1 we obtain the fact that any self-homomorphism of a minimal query is an automorphism.

For the rest of this section we shall assume that both Q and Q' are minimal fan-out free queries. In light of Lemma 5.1 we may also assume that they have the same number of NDV's and conjuncts. The basic outline of our equivalence algorithm is as follows:

For each conjunct $c \in C_Q$ we construct a forest of labelled trees $Forest[c] = \{Tree[c,c'] : c' \in C_Q,\ c' \text{ covers } c, \text{ and } c' \neq c\}$, where $Tree[c,c']$ is a labelled breadth-first spanning tree of the implication closure of $\{<c,c'>\}$ in the implication graph $G[Q]$. Similar forests are constructed for each $c \in C_{Q'}$. We then make use of the following two lemmas (to be proved later). For brevity we use the term *TI* to stand for *labelled forest isomorphism*, and use the symbol $\cong$ to denote both tree and query isomorphism.

Lemma 5.2. If $f:Q \rightarrow Q'$ is a query isomorphism, then $Forest[c] \cong Forest[f(c)]$ for every conjunct $c \in C_Q$.

Lemma 5.3. Suppose $c_1, c_2 \in C_Q$ and $Forest[c_1] \cong Forest[c_2]$. Then there is a query automorphism $f:Q \rightarrow Q$ such that $f(c_1) = c_2$.

These two lemmas yield the following fundamental theorem.

Theorem 5.1 Suppose $c \in C_Q$, $c' \in C_{Q'}$, and $Forest[c] \cong Forest[c']$. Then there is a query isomorphism $f:Q \rightarrow Q'$ if and only if there is a query isomorphism $f':Q \rightarrow Q'$ with $f(c) = c'$.

Proof. Suppose there is a query isomorphism $f:Q \rightarrow Q'$. If $f(c) = c'$ we are done, so suppose $f(c) = c''$. Note that this implies that c and c'' cover each other. Since c and c' cover each other by hypothesis, it thus must be the case that c' and c'' cover each other. By Lemma 5.2, $f(c) = c''$ implies that there is a tree isomorphism $t:Forest[c] \rightarrow Forest[c'']$. Since by assumption $Forest[c] \cong Forest[c']$, we conclude that there is a tree isomorphism $t':Forest[c'] \rightarrow Forest[c'']$. By Lemma 5.3 we thus conclude that there is a query automorphism $f':Q \rightarrow Q'$ with $f'(c'') = c'$. The composition of f followed by f' yields our desired query isomorphism. □

Our equivalence algorithm can thus proceed as follows.

1. Set $Q^* = Q$, $Q^{**} = Q'$, and $f = \varnothing$ (the empty symbol mapping).
2. While Q^* contains a conjunct c that contains a NDV, do the following:

   If there is no conjunct $c' \in C_{Q^{**}}$ such that $Forest[c] \cong Forest[c']$, then halt: Q and Q' are not equivalent by Lemma 5.2. Otherwise, let c^* be such a c', and extend the definition of f to include $f(c[j]) = c^*[j]$ for every j, $1 \leq j \leq Length(c)$ such that $c[j]$ is a NDV.

   If f fails to be either well-defined or one-one, halt: the queries are inequivalent. Otherwise, for each NDV u added to the domain of f, replace u and $f(u)$ in both Q^* and Q^{**} by a new unique constant symbol u^*.
3. At this point, Q^* no longer contains any NDV's. If the sorted lists of conjuncts for Q^*

and Q^{**} are identical, then the constructed f will correspond to a query isomorphism; otherwise, no such isomorphism exists (Theorem 5.1 guarantees that if there is a query isomorphism, the lists will be identical).

We now present the details of the construction of *Tree*$[c,c']$ and the proofs of Lemmas 5.2 and 5.3. We assume without loss of generality that c and c' are conjuncts of query Q. The tree T = *Tree*$[c,c']$ need not actually be a labelled breadth-first spanning tree for *all* of the implication closure of $\{<c,c'>\}$ in $G[Q]$, only for part of it. The basic idea is to start with the conjunct-pair vertex $<c,c'>$ as root, and expand the tree until either (a) the implication closure is completely spanned or (b) there is evidence that the set of conjunct-pair vertices in the implication closure of $\{<c,c'>\}$ does not induce a query automorphism.

There will actually be two labels associated with each vertex v of T, a *hidden label*, $H(v)$ and a *visible label*, *label*(v). The hidden labels are used in our construction of T and in our proofs, but are not considered when testing for tree isomorphisms. The hidden label of a vertex v is simply its name in $G[Q]$, i.e., $H[<c,c'>] = <c,c'>$ and $H[<y,z>] = <y,z>$. We shall use $H_1[v]$ and $H_2[v]$ to denote the first and second components of the hidden label of v, respectively.

The visible labels are the ones used in testing for tree isomorphism. The visible label *label*$[v]$ for a symbol-pair vertex v is of the form $<i,A>$, where i, a column index, is the *local address* and A is the *mapping conflict indicator*. The visible label for a conjunct-pair vertex v is of the form $<[R,j],P,A>$. Here $[R,j]$, with R being a relation name and j being a column index, is the local address. The second component, P, is the *profile*, and the third, A, is again the mapping conflict indicator. The visible labels are assigned so that every vertex v has a unique global address $a(v)$, this being the sequence of visible labels for all the vertices on the path from the root of T up to and including v.

The local address for a child u of v is assigned based on the reason for the adjacency of $H[u]$ and $H[v]$ in $G[Q]$. If $H[u] = <y,z>$ and $H[v] = <c_1,c_2>$, then the local address for u as a child of v is the least column index i such that $c_1[i] = y$ and $c_2[i] = z$ (such an index must exist by the definition of implication graph). If $H[u] = <c_1,c_2>$ and $H[v] = <y,z>$, then the local address $[R,j]$ for u has $R = R(c_1) = R(c_2)$ and j the least index such that $c_1[j] = y$ and $c_2[j] = z$. If u is the root of *Tree*$[c,c']$ and hence has no parent, its local address is $[R(c),0]$ by convention.

Note that although no conjunct-pair vertex can have two children with the same local address in *Tree*$[c,c']$, a symbol-pair vertex may have a number of conjunct-pair children with the same local address. One purpose of the profile is to distinguish between conjunct-pair children with the same local address. The profile is present in *label*$[v]$ for all conjunct-pair vertices v, and is determined entirely by $H_1[v]$. If $H_1[v] = c_1$ then the profile of v, which we shall denote by *profile*$[c_1]$), is a list of the form $<[J_1,b_1],[J_2,b_2],...,[J_k,b_k]>$, where the J_i's are sets forming a partition of the integers in $\{h : 1 \leq h \leq Length(c_1)\}$, and the b_i's are either DV's, constants, or the special symbol $\varnothing$. Each set J_i is a maximal set of indices h such that $c_1[h]$ has the same value for all $h \in J_i$. The entry b_i is this common value if that value is either a DV or a constant. If, however, the common value is a NDV, then b_i is by convention set to $\varnothing$ (our visible labels must be independent of the names of the NDV's). Thus we know that each J_i with $b_i = \varnothing$ represents a different NDV, but not which one. Within *profile*$[c_1]$ the sets are ordered according to their minimal elements, and each set has its members presented in increasing order.

If a vertex of *Tree*$[c,c']$ has a mapping conflict indicator other than 0, we say that that vertex has a *mapping conflict*. This will mean that $\{<c,c'>\}$ cannot be extended to a query automorphism. There will be at most one vertex in T with a mapping conflict (as soon as such a vertex is encountered in our tree generation process, we halt and declare T complete). If a symbol-pair vertex v has a mapping conflict, its mapping conflict indicator A is the global address of an earlier symbol-pair vertex u such that $H_1[u] = H_1[v]$ and $H_2[u] \neq H_2[v]$ (the fact that T contains such vertices means that $\{<c,c'>\}$ cannot extend to a well-defined query

automorphism).

If a conjunct-pair vertex v has a mapping conflict, there are three possibilities: the mapping conflict indicator for v can either be "1", "2", or the global address of some earlier conjunct-pair vertex. If the mapping conflict indicator for v is equal to 1, this means that $H[v] = <c'',\theta>$ for some conjunct c'', in which case $\{<c,c'>\}$ clearly cannot be extended to a query automorphism. If the mapping conflict indicator for v is equal to 2, this means that there are at least two conjunct-pair vertices that could be children of parent(v) with same visible label. The presence of two such vertices clearly destroys the uniqueness of global addresses, but as we shall see, it also means that $\{<c,c'>\}$ cannot be extended to a one-one self-homomorphism (and hence not to a query automorphism, since Q is minimal). If the mapping conflict indicator for v is the global address for an earlier conjunct-pair vertex u, this means that $H_1[u] = H_1[v]$ and $H_2[u] \neq H_2[v]$, and so again $\{<c,c'>\}$ cannot be extended.

This will be explained more fully as we now present an explicit procedure for constructing *Tree* $[c,c']$. We procede by levels. Odd levels contain conjunct-pair vertices and even levels contain symbol-pair vertices. The first level consists solely of the root, a single conjunct-pair vertex v with $H[v] = <c,c'>$ and *label* $[v] = <[R(c),0],$*profile* $[c],0>$. If all the vertices on level I, I odd, have been generated and the construction of *Tree* $[c,c']$ has not yet been terminated, the generation of the vertices of level $I+1$ proceeds according to procedure EVENLEVEL. (In the following a vertex v of *Tree* $[c,c']$ is *unprocessed* if we have not as yet attempted to generate its children; a vertex $<c_1,c_2>$ (or $<y,z>$) in $G[Q]$ is *unspanned* if there is as yet no vertex in *Tree* $[c,c']$ with $<c_1,c_2>$ (or $<y,z>$) as its hidden label).

Procedure EVENLEVEL

begin

Lexicographically order the vertices of level I according to their global addresses.

While there remains an unprocessed vertex on level I, let v be the lexicographically first such vertex and do the following:

begin

Let $H[v] = <c_1,c_2>$ and for each unspanned $<y,z>$ that is adjacent to $<c_1,c_2>$ in $G[Q]$, ordered by their potential local address (the least i such that $c_1[i]=y$ and $c_2[i]=z$), do the following:

begin

Create a new symbol-pair vertex u of *Tree* $[c,c']$, with v as parent, with i as local address, and with $H[u] = <y,z>$. If there is any already-generated vertex w with $H_1[w] = y$ (there can be at most one), then let A be the global address of w, set *label* $[u] = <i,A>$, and halt the generation of *Tree* $[c,c']$ (the tree is complete since we have encountered a mapping conflict - see Figure 4). Otherwise, set *label* $[u] = <i,0>$ and continue.

end

end

end EVENLEVEL

If all the vertices on level I, I even, have been generated and the construction of *Tree* $[c,c']$ has not yet been terminated, then we generate the vertices of level $I+1$ using the procedure ODDLEVEL:

procedure ODDLEVEL

begin

Lexicographically order the vertices of level I according to their global addresses, deleting all those v such that $H[v]$ either (a) has the same first and second components or (b) equals $<y,z>$, where all the vertices adjacent to $<y,z>$ in $G[Q]$ have the same first component (in other words, deleting all those v which correspond to symbol-pair vertices in $G[Q]$ that do not

imply anything).

While there remains an unprocessed vertex on level I, let v be the lexicographically first such vertex, and do the following:

begin
Let $H[v] = \langle y,z \rangle$. If $\langle y,z \rangle$ is adjacent in $G[Q]$ to a vertex with θ as second component, do the following:

begin
Let $D = \{c_1 \in C_Q$: there is a vertex $\langle c_1,\theta \rangle$ adjacent to $\langle y,z \rangle$ in $G[Q]\}$. For each $c_1 \in D$, let $A[c_1]$, the potential local address of c_1, be $[R,j]$, where $R = R(c_1)$ and j is the least index such that $c_1[j] = y$ and for no c_2 with $c_2[j] = z$ does c_2 cover c_1 (such a j must exist by the definition of implication graph). Then create a new vertex u as a child of v with $H[u] = \langle c^*,\theta \rangle$ and *label*$[u] = \langle A[c^*]$,*profile*$[c^*],1 \rangle$ where c^* is chosen to be a lexicographically minimum member of D with respect to the ($A[c^*]$,*profile*$[c^*]$) pair (See Figure 5). Since a mapping conflict has been found, halt and terminate the construction of *Tree*$[c,c']$.
end

Otherwise, all neighbors of $\langle y,z \rangle$ have distinct first components. For each potential local address $[R,j]$ of an unspanned neighbor of $\langle y,z \rangle$ in $G[Q]$, in lexicographic order, do the following:

begin
Let $D = \{\langle c_1,c_2 \rangle : \langle c_1,c_2 \rangle$ is an unspanned conjunct-pair vertex adjacent to $\langle y,z \rangle$ in $G[Q]$ with $R(c_1)=R$, $c_1[j]=y$, and $c_2[j]=z\}$. For each $\langle c_1,c_2 \rangle \in D$, in lexicographic order by values of *profile*$[c_1]$ do the following:

begin
Create a new vertex u as a child of v with $H[u] = \langle c_1,c_2 \rangle$ and construct its visible label as follows:

begin
If there is a $\langle c_3,c_4 \rangle \in D$, $\langle c_3,c_4 \rangle \neq \langle c_1,c_2 \rangle$ with *profile*$[c_3]$ = *profile*$[c_1]$, then the fact that $\langle c_1,c_2 \rangle$ preceded $\langle c_3,c_4 \rangle$ in our processing of D is only due to whatever arbitrary tie-breaking procedure we used to order conjunct-pairs in D whose first components had the same profile. We could just as well have had $\langle c_3,c_4 \rangle$ first and $H[u]$ would then have been assigned the value $\langle c_3,c_4 \rangle$. This is a mapping conflict of type 2 (See Figure 6). Set *label*$[u] = \langle [R,j]$,*profile*$[c_1],2 \rangle$ and terminate the construction of *Tree*$[c,c']$.

If there is an already-generated vertex w of *Tree*$[c,c']$ with $H_1[w] = c_1$ (there can be at most one), then let A be the global address of w, set *label*$[u] = \langle [R,j]$,*profile*$[c_1],A \rangle$, and halt the generation of *Tree*$[c,c']$.

Otherwise, set *label*$[u] = \langle [R,j]$,*profile*$[c_1],0) \rangle$ and continue.
end
end
end
end
end ODDLEVEL

The construction of *Tree*$[c,c']$ continues until terminated as above or until an entire level is processed without creating any new vertices.

It should be clear that this construction yields a *Forest*$[c]$ whose topology and labelling is totally independent of the names of the NDV's in Q and of the order of the conjuncts in C_Q.

This information is not used explicitly in our labellings, and whenever ties are broken lexicographically, the sorting is done in terms of relation names, column indices, DV's, constant names, profiles, and special symbols such as θ and $\varnothing$, none of which depend on the NDV's or conjunct ordering, and all of which would be present in any query equivalent to Q. Thus the proof of Lemma 5.2, that if Q Q' then the corresponding conjuncts have isomorphic forests, would only involve a straightforward (and tedious) induction, which we shall omit.

The proof of Lemma 5.3 is much more interesting, and will proceed via a sequence of sublemmas.

Lemma 5.3.1. If *profile* $[c_1]$ = *profile* $[c_2]$, then c_1 and c_2 cover each other, and any conjunct that covers one must cover the other.

Proof. The fact that c_1 and c_2 have the same profile means that they are the identical up to a renaming of their NDV's. Hence there is a symbol mapping f that sends c_1 to c_2 and a symbol mapping f' that sends c_2 to c_1, and so each covers the other. The second half of the lemma follows from the fact that the "covering" relation is transitive. □

Lemma 5.3.2. If there is a conjunct-pair vertex u in *Tree* $[c,c']$ with a mapping conflict indicator of 2, then the implication closure of $\{<c,c'>\}$ in $G[Q]$ contains two conjunct-pair vertices $<c_1,c_2>$ and $<c_3,c_2>$, $c_1 \neq c_2$ and $c_2 \neq \theta$, either of which could have been assigned as the value of $H[u]$ by procedure ODDLEVEL.

Proof. Let $H[u] = <c_1,c_2>$. By the operation of ODDLEVEL in assigning an mapping conflict indicator of 2, u was the child of some symbol-pair vertex $<y,z>$, all of whose neighbors had distinct first components, and there was a second conjunct-pair $<c_3,c_4>$ that could also have been assigned as $H[u]$ with $c_2 \neq \theta \neq c_4$. We now show $c_2 = c_4$.

Suppose $c_2 \neq c_4$. Since c_1 and c_3 have the same profiles and c_4 covers c_3, we would have by Lemma 5.3.1 that c_4 also covers c_1. Thus both c_2 and c_4 would cover c_1, and hence $<c_1,c_2>$ and $<c_1,c_4>$ would be distinct neighbors of $<y,z>$ in $G[Q]$. However, $<y,z>$ is adjacent to only one conjunct-pair with c_1 as first component, and so $c_2 = c_4$ and $<c_1,c_2>$ and $<c_3,c_2>$ are the desired conjunct-pairs. □

Lemma 5.3.3. For all conjunct-pairs $<c,c'>$ of Q such that c' covers c and $c \neq c'$, if *Tree* $[c,c']$ contains no vertex with a mapping conflict, then there is a query automorphism $f: Q \rightarrow Q$ with $f(c) = c'$.

Proof. If *Tree* $[c,c']$ contains no vertex with a mapping conflict, then it must span the entire implication closure of $\{<c,c'>\}$. Let V' be the set of conjunct-pair vertices in this closure. We claim that V' obeys the partial homomorphism property. Condition (C1) is satisfied since if V' had contained a vertex with θ as second component, this would have given rise to a mapping conflict of type 1 for a conjunct-pair vertex in *Tree* $[c,c']$. Condition (C2) is satisfied since if the set of symbol-pair vertices in V' did not obey the partial symbol mapping property, it would contain vertices $<y,z>$ and $<y,z'>$, $z \neq z'$, and the second of these to be encountered in the generation of *Tree* $[c,c']$ would have had a mapping conflict (unless the generation of *Tree* $[c,c']$ had already been terminated on account of some earlier mapping conflict).

Since V' obeys the partial homomorphism property and $<c,c'> \in V'$, there is a self-homomorphism of Q that sends c to c' by Theorem 3.3. Since Q is minimal, this self-homomorphism is the desired query automorphism. □

Lemma 5.3.4. If there is a tree isomorphism t:*Forest* $[c] \rightarrow$ *Forest* $[c']$ and $c \neq c'$, then for each vertex v in *Tree* $[c,c']$, $H_2[v] = H_1[t(v)]$.

Proof. Let *Tree* $[c',c'']$ be the image under t of *Tree* $[c,c']$. We proceed by induction on the levels of the two trees. In each the first level consists of a single root vertex. If v is the root vertex in *Tree* $[c,c']$, then $t(v)$ must be the root vertex in *Tree* $[c',c'']$. Hence we have $H[v] = <c,c'>$ and $H[t(v)] = <c',c''>$, and the lemma holds for the first level.

Suppose it holds for *parent* (u) and t(*parent* (u)). We show it holds for u. First note that t(*parent* (u)) = *parent* $(t(u))$, since t is an isomorphism. We now split into cases, depending on whether u is a symbol-pair or a conjunct-pair vertex.

Suppose u is a symbol-pair vertex. By hypothesis there exist c_1, c_2, and c_3 such that $H[parent(u)] = <c_1,c_2>$ and $H[parent(t(u))] = <c_2,c_3>$. Suppose $H[u] = <y,z>$ and $H[t(u)] = <y',z'>$. Both u and $t(u)$ must have the same local address, say i. This means that $c_1[i]=y$ and $c_2[i]=z$, and also that $c_2[i]=y'$ and $c_3[i]=z'$. But then $z=y'$ and so $H_2[u] = H_1[t(u)]$, as desired.

Suppose u is a conjunct-pair vertex. This is a more complicated case. By hypothesis there exist y, z, and z' such that $H[parent(u)] = <y,z>$ and $H[parent(t(u)] = <z,z'>$. Suppose $H[u] = <c_1,c_2>$ and $H[t(u)] = <c_3,c_4>$. Since t preserves visible labels, we must have $profile[c_1] = profile[c_3]$. Thus c_3 covers c_1 by Lemma 5.3.1. Moreover, $<c_1,c_3>$ would be adjacent to $<y,z>$ in $G[Q]$: the fact that t preserves visible labels means that u and $t(u)$ share the same local address $[R,j]$, which implies that $c_1[j] = H_1[parent(u)] = y$ and $c_3[j] = H_1[parent(t(u))] = z$. We cannot have $c_2 = \theta$, because such a vertex could exist only if there were no conjunct c_3 that covered c_1 and had $c_3[j]=z$. However, $parent(u)$ can have a child with a hidden label of $<c_1,c_2>$, $c_2 \neq \theta$, only if $<y,z>$ is adjacent to no vertex in $G[Q]$ with θ as second component. Since Q is fan-out free, this means that all neighbors of $<y,z>$ in $G[Q]$ have distinct first components. The conjunct-pair $<c_1,c_2>$ is a neighbor because $<c_1,c_2>$ is the hidden label of a vertex in $Tree[c,c']$ whose parent has hidden label $<y,z>$. We have just argued that $<c_1,c_3>$ must also be a neighbor. Since these two conjunct-pairs have the same first component, they cannot be distinct neighbors, and so we must have $c_2 = c_3$, as claimed. □

Proof of Lemma 5.3. Suppose that $t: Forest[c] \rightarrow Forest[c']$ is a tree isomorphism. We must show that there is a query automorphism $f: Q \rightarrow Q$ such that $f(c) = c'$. By Lemma 5.3.3, this reduces to showing that $Tree[c,c']$ contains no vertex with a mapping conflict, since c and c' must have the same profile if their forests are isomorphic, and hence must cover each other by Lemma 5.3. We shall show that each of the four possible mapping conflicts is impossible.

(1) Suppose there is a symbol-pair vertex u in $Tree[c,c']$ with a non-zero mapping conflict indicator A, and let v be the vertex with global address A in $Tree[c,c']$. We must have $H[u] = <y,z>$ and $H[v] = <y,z'>$ for some $z' \neq z$. Now consider $t(u)$ and $t(v)$. By Lemma 5.3.4, we must have $H_1[t(u)]=H_2[u]=z$ and $H_1[t(v)] = H_2[v] = z'$. But then $H_1[t(u)] \neq H_1[t(v)]$, and so there cannot be a mapping conflict between $t(u)$ and $t(v)$, even though the fact that t preserves labels means that there must be. Hence we have a contradiction.

(2) Suppose there is a conjunct-pair vertex u with a mapping conflict of type 1. Then $H_2[u] = \theta$. But this means, by Lemma 5.3.4, that $H_1[t(u)] = \theta$, an obvious contradiction since θ can never occur as a first component.

(3) Suppose that there is a conjunct-pair vertex u with a mapping conflict of type 2. This means, in our construction procedure ODDLEVEL, $H[u]$ was chosen by an arbitrary tie-breaking rule from two conjunct-pairs with the same local addresses and profiles, and by Lemma 5.3.2 these pairs are of the form $<c_1,c_2>$ and $<c_3,c_2>$, $c_1 \neq c_3$. Because t preserves labels, $H[t(u)]$ must also have been chosen from two conjunct-pairs $<c_1',c_2'>$ and $<c_3',c_2'>$, $c_1' \neq c_3'$. But by Lemma 5.3.4, applied to the two alternative hidden labels for $t(u)$ in $Tree[c,c']$, we must also have both $c_1'=c_2$ and $c_3'=c_2$, so again we have a contradiction.

(4) The case where a conjunct-pair vertex u has a mapping conflict indicator which is the global address of some earlier conjunct-pair vertex is handled in the same way as case (1), leading to a similar contradiction.

Thus all possibilities for mapping conflicts are impossible and the desired query automorphism exists by Lemma 5.3.3. □

Since Lemma 5.3, together with Lemma 5.2, implies Theorem 5.1, we now know that our equivalence algorithm works. The remainder of this section is devoted to questions of its efficiency.

We first observe that labeled forest isomorphism can be tested in time proportional to the *size* of the forests, i.e., number of vertices in the two forests plus the total length of the labels

[1]. Since our basic algorithm performs at most $O(N^2)$ labeled forest isomorphism tests, where N is the number of conjuncts in Q or Q', we thus can bound the overall running time for the labeled forest isomorphism tests by placing a bound on the size of *Forest*$[c]$ for any $c \in C_Q$. This size is in turn bounded by N times the maximum size of any *Tree*$[c,c']$, c' covers c, and so it is this latter quantity that we shall examine in detail.

In what follows, let us identify each vertex of *Tree*$[c,c']$ with the value of its hidden label. In addition, we make the standard assumption that our computer word size is sufficient to be able to hold any relation name R, distinguished variable name x, constant z, or index i. Thus the only visible labels that require more than a constant amount of space are the conjunct-pair labels (because of their profiles) and the (at most) one label that has a global address for its mapping conflict indicator. For the purpose of asymptotic analysis, we can ignore the latter case, since a global address is merely a concatenation of other labels, and hence can at most double the total space required for labels.

We first examine the symbol-pair vertices of *Tree*$[c,c']$. All but possibly the last symbol-pair vertex of *Tree*$[c,c']$ have distinct NDV's as first components, and since the number of NDV's in Q is $O(n)$, where n is the sum of the lengths of the conjuncts in Q, this means that there are $O(n)$ symbol-pair vertices in *Tree*$[c,c']$. From this and the fact that each symbol-pair label (except possibly the last) requires only constant space, we conclude that the size of *Tree*$[c,c']$ attributable to symbol-pair vertices is $O(n)$.

The situation is only slightly more complicated for the conjunct-pair vertices. Discounting the last such vertex (should it have a mapping conflict), all conjunct-pair vertices have distinct first components. Moreover, the only non-constant-sized part of the visible label for a conjunct-pair with c_1 as first component is *profile*$[c_1]$, whose length is proportional to *Length*(c_1). Thus the total number of conjunct-pair vertices is $O(N)$ and the total length of their labels is at most $O(n)$.

We conclude that the size of *Tree*$[c,c']$ is at most $O(n)$, and hence the overall time for the labeled forest isomorphism tests is $O(N^3n) = O(n^4)$ time. This will also be a bound on the running time for the entire algorithm, so long as each *Tree*$[c,c']$ can be *generated* in time $O(n)$.

Our general plan will be first to generate the implication graph for the current query, and then use this to generate the *Tree*$[c',c'']$s. In our overall algorithm the original queries are modified at most N times each, so we need generate at most $2N$ implication graphs, at $O(n^2)$ time each (as shown in the previous section). Since we are assuming we generate $O(n^3)$ *Tree*$[c',c'']$s, this works out to only a constant amount of time for each. In generating the implication graphs, we thus can afford to do some extra work. In particular, we order the adjacency lists for the symbol-pair vertices so that all conjunct-pair vertices with θ as second component precede all conjunct-pair vertices with a conjunct as second component. We now argue that the time for generating an individual *Tree*$[c,c']$ is $O(n)$.

It is not difficult to see that the time for generating the children of any conjunct-pair vertex $<c_1,c_2>$ in *Tree*$[c,c']$ will be proportional to *Length*(c_1). Such a vertex $<c_1,c_2>$ is adjacent to at most *Length*(c_1) neighbors. For each such neighbor $<y,z>$ we first (temporarily) delete $<c_1,c_2>$ from the adjacency list for $<y,z>$ in $G[Q]$, so that in generating the children of $<y,z>$ we will not have to look at $<c_1,c_2>$, which is already in the tree. Then we add $<y,z>$ to the tree if it is not already there. The total time is proportional to *Length*(c_1) if adjacency lists are stored as linked lists in $G[Q]$, with special fields so that temporary deletions can be made using the trick from [1] that allows us to dispense with initialization. (We also use this trick to allow us to test in constant time whether a new symbol-pair vertex has a mapping conflict).

For a symbol-pair vertex, none of whose children has a mapping conflict, it is not difficult to see that the time is proportional to the size of the children in *Tree*$[c,c']$. By our deletion trick above, we know we only have to look at those children that are not already in the tree, and each label can be generated in time proportional to its length (there is, in fact, no need to

compute the profile, since profiles for all conjuncts can be generated in a preprocessing step). What sorting needs to be done can be done in time proportional to the lengths of the profiles using standard lexicographic sorting techniques [1], and mapping conflicts with earlier vertices can once more be checked with our indexing trick.

The only potential difficulty comes when generating the children of a symbol-pair vertex $<y,z>$ which *has* a child with a mapping conflict, since we might end up looking at all the neighbors of $<y,z>$ and then generating only one child. However, note that, since Q and all the queries derived from it are fan-out free, $<y,z>$ is either adjacent to a $<c_1,\theta>$ vertex or else all of its neighbors have distinct first components.

In the first case, our generation of the implication closure insures that the $<c_1,\theta>$ vertices will head the adjacency list for $<y,z>$ in the implication closure. In order to generate the only child of $<y,z>$ (and last vertex of $Tree[c,c']$), we must order these vertices lexicographically by (local address, profile) pairs. This ordering can be done in time $O(n)$, since there is at most one vertex to consider with any conjunct as first component, local addresses are of constant length, and so the total length of relevant profiles is proportional to the sum of the lengths of the conjuncts.

In the latter case, the total work involved in generating all the children of $<y,z>$ up to the one with the mapping conflict will also be $O(n)$, since it will again be proportional to the sum of the lengths of the first components. If a global address must be found for the mapping conflict indicator, this will again be doable in time proportional to the size of the already-generated tree.

Thus the time for generating $Tree[c,c']$ will be proportional to its size, which is $O(n)$, plus a final $O(n)$ for the last vertex, or $O(n)$ in total, and the overall running time of the equivalence testing algorithm is $O(n^4)$.

6. CONCLUDING REMARKS

We have extended the class of queries for which minimization and equivalence testing can be performed in polynomial time to a new class, the "fan-out free" queries, that is a generalization of classes considered in [2,3,7,8], and allows for queries which are not restricted to untyped variables and hence can ask for transitive information about databases. The running times for our algorithms - $O(n^3)$ and $O(n^4)$ respectively - are the same as the corresponding running times in the original papers [2,3] on "simple" queries. One direction for future research would be to try to improve the running times of these algorithms, as did [7] for the simple query algorithms. For example, in the equivalence testing problem, it would seem possible to do a certain amount of "pruning" to our $Tree[c,c']$s and still get our algorithm to work, although whether such an approach will yield an asymptotic improvement in worst-case running time is more problematic.

Another direction for further research would be to further extend the class of queries that can be handled. The minimization algorithms of [7] for the class of typed queries in which no conjunct has more than one occurence of a repeated variable (the queries obeying restriction (2) of Section 2), would seem to extend more-or-less directly to the untyped case, yielding $O(n^2)$ algorithms. Although we have already remarked that queries in this class cannot ask any questions that a fan-out free query couldn't also ask, there are queries in the class that are not fan-out free. One might hence be able to construct a combined minimization algorithm, which not only handled all queries that either obeyed restriction (2) or were fan-out free, but also could deal with queries which were in neither class, but partook of certain aspects of both. Whether one could also obtain a polynomial time equivalence algorithm for such an expanded class appears to be a harder question.

One also might look to see what can be done when the restriction to *conjunctive* queries is removed. Sagiv and Yannakakis [8] examined this question in the case of typed queries. What can be said when untyped variables are allowed? Also, what about databases obeying more complicated dependencies than mere functional dependencies?

Finally, we note that the techniques of Section 5 may have implications beyond database theory. Essentially what we did in this section was show that a special case of the graph isomorphism problem could be solved by constructing canonical labelled forests and using known algorithms to test for isomorphism between *them*. It may well be that there are other isomorphism or equivalence problems which, although equivalent to graph isomorphism in general, have meaningful special cases which can be solved in an analogous fashion.

REFERENCES

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA (1974), 84-86.
2. A.V. Aho, Y. Sagiv, and J.D. Ullman, "Equivalences among relational expressions," *SIAM J. Computing,* 8 (1979), 218-246.
3. A.V. Aho, Y. Sagiv, and J.D. Ullman, "Efficient optimization of a class or relational expressions," *ACM Trans. on Database Systems,* 4 (1979), 435-454.
4. A.K. Chandra and P.M. Merlin, "Optimal implementation of conjunctive queries in relational data bases," *Proc. 9th Ann. ACM Symp. on Theory of Computing,* Association for Computing Machinery, New York (1977), 77-90.
5. E.F. Codd, "A relational model of data for large shared data banks," *Commun. ACM,* 13 (1970), 377-387.
6. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman and Co., San Francisco (1979).
7. Y. Sagiv, "Quadratic algorithms for minimizing joins in restricted relational expressions," Report UIUCDCS-R-79-992, Department of Computer Science, University of Illinois, Urbana, IL (1979).
8. Y. Sagiv and M. Yannakakis, "Equivalences among relational expressions with the union and difference operators," *J. Assoc. Comput. Mach.,* 27 (1980), 633-655.
9. M. Yannakakis and C. H. Papadimitriou, "Algebraic dependencies," *Proc. 21st Symp. on Foundations of Computer Science,* IEEE Computer Society, Long Beach, CA (1980), 328-332.

Parent	Child
George III	Adolphus
Victoria	Alice
Albert	Alice
Adolphus	Mary
George V	George VI
Mary	George VI
Elizabeth	Elizabeth II
George VI	Elizabeth II
Victoria	Edward VII
Edward VII	George V

Figure 1. Relation R_1 with scheme <Parent,Child>. If $R_1(x,y)$, then x is the parent of y.

Parent	Child
George III	George III
Victoria	Victoria
Albert	Albert
Adolphus	Adolphus
George V	George V
Mary	Mary
Elizabeth	Elizabeth
George VI	George VI
Edward VII	Edward VII
Elizabeth II	Elizabeth II

Figure 2. Relation R_2 with scheme <Parent,Child>. If $R_2(x,y)$, then x is the same person as y.

Base Relation Scheme R_1 = <Parent, Child>

Target Relation Scheme R_Q = <Great-great-grandparent, Child>

$X_Q = \{x_1, x_2\}$, $Y_Q = \{y_1, y_2, y_3\}$

$C_Q = \{R_1(x_1, y_1), R_1(y_1, y_2), R_1(y_2, y_3), R_1(y_3, x_2)\}$

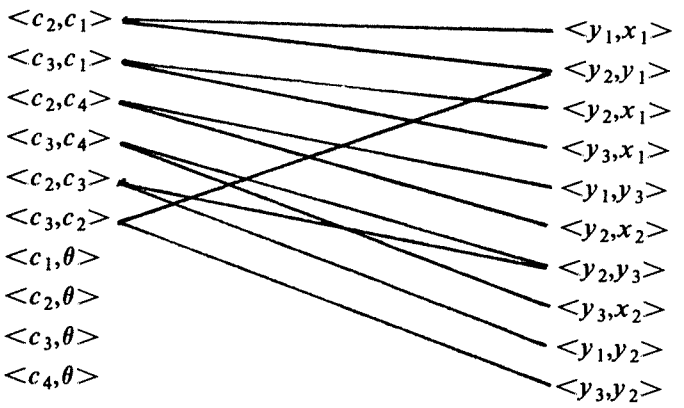

Figure 3. A query Q and its implication graph G_Q.

Tree$[c_1,c_2]$	$c_1=R_1(u)$, $c_2=R_1(v)$ $c_3=R_2(u,y)$, $c_4=R_2(v,z)$ $c_5=R_3(u,y)$, $c_6=R_3(v,x)$

(vertices represented by "hidden label : visible label")

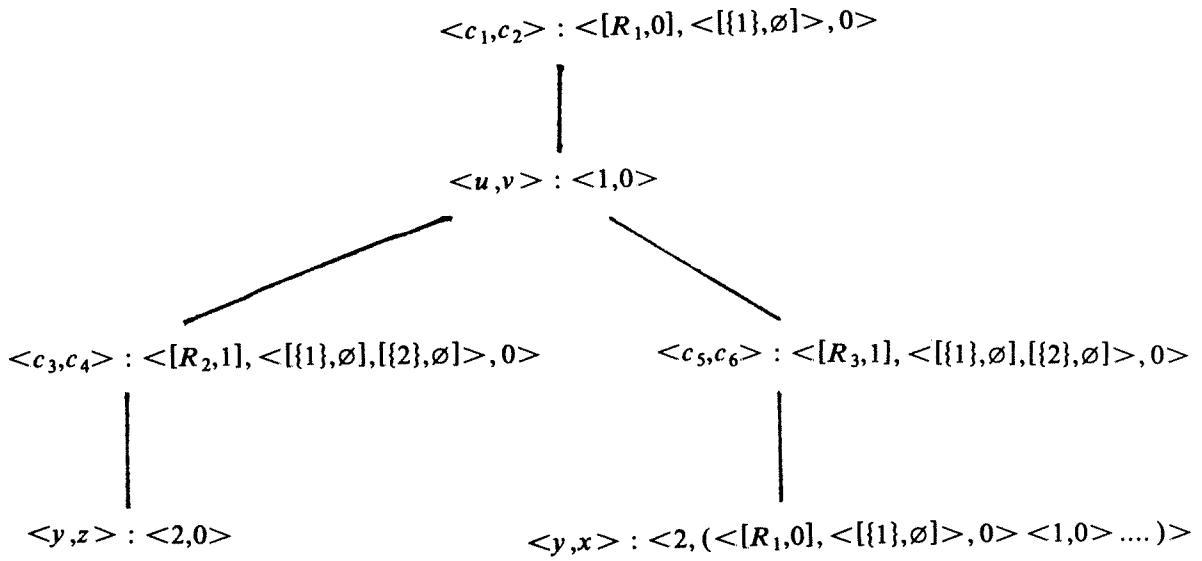

Figure 4. Example of a symbol-pair mapping conflict. The full MCI is the concatenation of all visible labels on the path from $<c_1,c_2>$ through $<y,z>$.

$c_1 = R_1(u),\ c_2 = R_1(v),\ c_3 = R_2(u,5),\ c_4 = R_2(v,10)$

Tree$[c_1,c_2]$

$<c_1,c_2> : <[R_1,0], <[\{1\},\varnothing]>, 0>$

$<u,v> : <1,0>$

$<c_3,\theta> : <[R_2,1], <[\{1\},\varnothing],[\{2\},5]>, 1>$

Figure 5. Example of a conjunct-pair mapping conflict in which the MCI has value 1.

$c_1 = R_1(u),\ c_2 = R_1(v),\ c_3 = R_2(10,u,10,y),\ c_4 = R_2(10,u,10,z),\ c_5 = R_2(10,v,10,x)$

Tree$[c_1,c_2]$

$<c_1,c_2> : <[R_1,0], <[\{1\},\varnothing]>, 0>$

$<u,v> : <1,0>$

$<c_3,c_5> : <[R_2,2], <[\{1,3\},10],[\{2\},\varnothing],[\{4\},\varnothing]>, 2>$

Figure 6. Example of a conjunct-pair mapping conflict in which the MCI has value 2. For simplicity, we have used a non-minimal query as our example. A minimal query with the same mapping conflict can be otained by adding extra conjuncts to insure that the NDVs cannot map to each other.