SIAM J. COMPUT.
Vol. 8, No. 2, May 1979

©1979 Society for Industrial and Applied Mathematics
0097-5397/79/0802-0009 $01.00/0

EQUIVALENCES AMONG RELATIONAL EXPRESSIONS*

A. V. AHO†, Y. SAGIV‡ AND J. D. ULLMAN¶

Abstract. Many database queries can be formulated in terms of expressions whose operands represent tables of information (relations) and whose operators are the relational operations select, project, and join. This paper studies the equivalence problem for these relational expressions, with expression optimization in mind. A matrix, called a tableau, is proposed as a natural representative for the value of an expression. It is shown how tableaux can be made to reflect functional dependencies among attributes. A polynomial time algorithm is presented for the equivalence of tableaux that correspond to an important subset of expressions, although the equivalence problem is shown to be NP-complete under slightly more general circumstances.

1. Introduction. Codd's relational algebra is a high-level query language in which questions can be posed simply and succinctly [9], [11]. Concepts from relational algebra have been incorporated into the design of several new database query languages [13].

Expressions in relational algebra manipulate tables of information (called relations) by means of high-level operations such as select, project, and join. A disadvantage of relational algebra as a query language is that the efficiency with which a query can be answered varies considerably with the manner in which the query is formulated. The very flexibility of the language makes it easy to express queries that are hard to implement or for which efficient implementations are hard to find. Consequently, a number of papers [17], [19], [20], [21], [23], [25] have considered transformations that "optimize" relational queries. Like most work in code "optimization," however, these transformations improve expressions under some cost criterion, but do not claim to produce an equivalent expression of least cost. Chandra and Merlin [8] show how to perform true optimization on a large class of queries, but their algorithm is exponential in the size of the query.

In this paper we consider the inherent computational complexity of determining whether two queries are equivalent, with an eye toward globally optimizing queries under a variety of cost measures. We restrict the relational algebra to include only the three operators: select, project, and join. We show that the optimization problem for even this restricted subset of relational algebra is computationally difficult (NP-complete).

We introduce tableaux, two-dimensional representations of queries. Tableaux may be viewed as a form of Zloof's "Query-by-Example" language [27] and also as a stylized notation for a subset of Chandra and Merlin's "conjunctive queries" [8]. The tableau immediately removes one objection (see [24], e.g.) to relational algebra as a query language, since tableaux are nonprocedural representations of queries in exactly the sense that relational calculus [9], [11] is nonprocedural.

We reduce the equivalence problem for queries to the analogous problem for tableaux. One advantage of the tableau approach is that it allows us to deal with functional dependencies mechanically, a feature not possessed by more direct techniques. We then show how to minimize the number of rows in a tableau, an operation that corresponds to minimizing the number of joins needed to evaluate a query. Since join is typically a very expensive operator to implement, this approach is a good "first crack" at reducing the cost of evaluating a query. Row minimization also serves to eliminate common subexpressions from a query.

* Received by the editors March 23, 1978. This work was supported in part by the National Science Foundation under Grant MCS-76-15255.

† Bell Laboratories, Murray Hill, New Jersey, 07974.

‡ Princeton University, Princeton, New Jersey. Now at Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 6180.

¶ Princeton University, Princeton, New Jersey 08540.

Next we introduce "simple tableaux," a subclass of tableau for which we can show the equivalence and optimization problems that were computationally difficult for general tableaux are now tractable. Although the set of queries having simple tableaux is a proper subset of the set of relational expressions, we nevertheless feel that most practical queries that contain only selects, projects, and joins can be represented by simple tableaux. We conclude the paper with a discussion of some remaining problems.

2. Basic definitions. In this section we define our restricted subclass of relational expressions. We also show that there are several possible definitions of expression equivalence.

2.1. Relation schemes and relations. We assume the data are stored in a set of two-dimensional tables called *relations*. The columns of a table are labeled by distinct *attributes* and the entries in each column are drawn from a fixed *domain* for that column's attribute. For the purposes of this paper we assume the ordering of the attributes of a table is unimportant. Each row of a table is a mapping from the table's attributes to their respective domains. A row is often called a *tuple* or *record*. If r is a relation that is defined on a set of attributes that includes A, and if μ is a tuple of r, then $\mu(A)$ is the value of the A-component of μ.

A *relation scheme* is the set of attributes labeling the columns of a table. When there is no ambiguity, we shall use the relation scheme itself as the name of the table. A relation is just the "current value" of a relation scheme. The relation is said to be *defined on* the set of attributes of the relation scheme.

Example 1. Suppose we have the two relation schemes PAT and PR, representing two tables, one with columns P, A, and T, the other with columns P and R. (P stands for Paper-number, A for Author, T for Title, R for Referee.) Figure 1 shows two relations that might be current values of these relation schemes.

P	A	T
1	Black	*All About Horses*
2	Brown	*All About Dogs*
3	Blue	*All About Cats*

P	R
1	Turtle
1	Snake
2	Turtle
3	Ox

FIG. 1. *Two tables.*

2.2. Dependencies. Often the values of entries in relations satisfy certain constraints. Functional [4], [9] and multivalued [7], [14], [15], [26] dependencies are examples of such constraints. In this paper we assume all dependencies are functional. Our theory carries over to multivalued dependencies as well, although an efficient equivalence test in that case is elusive.

A *functional dependency* is a statement $X \to Y$, where X and Y are sets of attributes. A relation r satisfies this functional dependency if and only if for all μ and ν in r the following condition holds: If $\mu(A) = \nu(A)$ for all A in X, then $\mu(B) = \nu(B)$ for all B in Y. That is, if two rows of r agree in the columns for X, then they must agree in the columns for Y. Note that if r satisfies a given set of dependencies, then it may also satisfy additional dependencies, e.g., if r satisfies $A \to B$ and $B \to C$, it also satisfies $A \to C$.

For a set of attributes X, we define X^*, the *closure* of X, as follows:

(1) $X \subseteq X^*$.

(2) If $Y \subseteq X^*$, and $Y \to Z$ is a given functional dependency, then $Z \subseteq X^*$.

(3) No attribute is in X^* unless it so follows from (1) and (2).

We write $X \overset{*}{\rightarrow} Y$ if $Y \subseteq X^*$. Essentially, $X \overset{*}{\rightarrow} Y$ means that the functional dependency $X \rightarrow Y$ is in, or can be derived from, the given set of dependencies. Two sets of dependencies are *equivalent* if, for all X, the set X^* is the same under either set of dependencies. It is well known that any set of dependencies is equivalent to a set in which each right side consists of a single attribute, and we henceforth assume all sets of functional dependencies are of this form.

2.3. Restricted relational expressions. In this paper we shall consider relational expressions in which the only operators are select, project, and (natural) join. The operands are relation schemes. The operators are defined as follows.

(1) *Select.* Let r be a relation on a set of attributes X, A an attribute in X, and c a value from the domain of A. Then the *selection* $A = c$, written $\sigma_{A=c}(r)$, is the set

$$\{\mu \mid \mu \text{ is in } r \text{ and } \mu(A) = c\}$$

that is, the subset of r having value c for attribute A.

(2) *Project.* Let r be a relation on a set of attributes X. Let Y be a subset of X. We define $\pi_Y(r)$, the *projection* of r onto Y, to be the relation obtained by removing all the components of the tuples of r that do not belong to Y and identifying common tuples. That is, $\pi_Y(r) = \{\nu \mid \nu$ has components for all and only the attributes of Y, and for some μ in r, $\nu(A) = \mu(A)$ for all A in $Y\}$.

For example, if r is the second relation of Fig. 1, then $\pi_P(r) = \{1, 2, 3\}$.

(3) *Join.* The join operator, denoted by $\bowtie$, permits two relations to be combined into a single relation whose attributes are the union of the attributes of the two argument relations. Let R_1 and R_2 be two relation schemes with current values r_1 and r_2. Then

$$r_1 \bowtie r_2 = \{\mu \mid \mu \text{ is a tuple with components for all and only the attributes in } R_1 \cup R_2, \text{ and there exist tuples } \nu_1 \text{ in } r_1 \text{ and } \nu_2 \text{ in } r_2, \text{ such that } \nu_1(A) = \mu(A) \text{ for all } A \text{ in } R_1 \text{ and } \nu_2(A) = \mu(A) \text{ for all } A \text{ in } R_2\}.$$

Example 2. If r_1 and r_2 are the two relations of Fig. 1, then $r_1 \bowtie r_2$ is the relation

P	A	T	R
1	Black	*All About Horses*	Turtle
1	Black	*All About Horses*	Snake
2	Brown	*All About Dogs*	Turtle
3	Blue	*All About Cats*	Ox

□

Even with these three simple operators we can pose a variety of interesting queries. Here are two examples that refer to the database in Fig. 1.

(1) The query "List the author of the paper *All About Dogs*" can be represented by the expression $\pi_A(\sigma_{T=\text{"All About Dogs"}}(PAT))$.

(2) "List the authors and titles of all papers refereed by Turtle" becomes $\pi_{AT}(\sigma_{R=\text{"Turtle"}}(PAT \bowtie PR))$.

With these operators we can also define Cartesian product (if in a join the sets of attributes for the two relations are made disjoint) and intersection (which is a special case of the natural join where the two relations are over the same set of attributes). The relational algebra of Codd [9], [11] includes other operators, and to make a "complete" set we would need to add union, set difference and selections involving arithmetic comparisons between two components of a tuple.

2.4. Expression values. The notion that a relation is the "value" of a relation scheme can be generalized to expressions. Let E be an expression with operand relation schemes $R_1, R_2, \cdots, R_k$. An *assignment* associates a relation r_i with each relation scheme R_i, $1 \leqq i \leqq k$. Given an assignment α of relations to relation schemes, the value of E, denoted $\nu_\alpha(E)$ or $\nu(E)$ if α is understood, is computed by applying operators to operands in the following natural way.

(1) If E is a single relation scheme R_i, then $\nu(E) = r_i$.

(2) (a) If $E = \sigma_{A=c}(E_1)$, then $\nu(E) = \sigma_{A=c}(\nu(E_1))$.

(b) If $E = \pi_X(E_1)$, then $\nu(E) = \pi_X(\nu(E_1))$.

(c) If $E = E_1 \bowtie E_2$, then $\nu(E) = \nu(E_1) \bowtie \nu(E_2)$.

We may also regard expression E as a function, mapping assignments of values for its operands to values for the expression. That is, if E is an expression with operands $R_1, R_2, \cdots, R_k$, we define $V(E)$ to be the mapping that sends each assignment α of relations $r_1, r_2, \cdots, r_k$ for $R_1, R_2, \cdots, R_k$ to $\nu_\alpha(E)$. Intuitively, two expressions E_1 and E_2 are equivalent if $V(E_1)$ and $V(E_2)$ are the same mapping. However, we may not wish to allow completely arbitrary sets of R_i's and r_i's. We have therefore isolated three distinct notions of equivalence, which we shall discuss in turn.

2.5. Algebraic equivalence. If we do not fix the R_i's, that is, allow each relation scheme to be a variable set of attributes, we obtain a notion of algebraic equivalence. For example, the commutative law of joins $R \bowtie S = S \bowtie R$ is true independent of R and S. It is not clear how the select operator can be brought into this framework, although the project operator π_X can be covered if we regard X as a variable set of attributes. We shall not discuss algebraic equivalence further in this paper.

2.6. Strong equivalence. We may, instead, regard each $R_1, R_2, \cdots, R_k$ as a relation scheme with a fixed set of attributes, and call two expressions E_1 and E_2 *strongly equivalent* if $V(E_1) = V(E_2)$ under this assumption. That is, we regard E_1 and E_2 as equivalent if they define the same mapping. Strong equivalence appears to be the notion underlying previous attempts at expression optimization, and is probably the notion with which most people would feel secure.

2.7. Weak equivalence. A variety of papers such as [1], [4], [7] have viewed a database as though a single universal relation exists at each instant of time. In this framework we restrict assignments of values to relation schemes $R_1, R_2, \cdots, R_k$ by insisting that there be some relation I on the set of attributes $\bigcup_{i=1}^{k} R_i$ such that the value r_i assigned to R_i is $\pi_{R_i}(I)$. We call such a relation I an *instance of the universe*, or just an *instance*. If $\nu_\alpha(E_1) = \nu_\alpha(E_2)$ for all assignments α obtained in this way from an instance, then we say E_1 and E_2 are *weakly equivalent*, and write $E_1 \equiv E_2$.

The notion of weak equivalence is also well motivated. It is essential when we deal with equivalences between expressions whose operands are different relation schemes. For example, it allows the treatment of lossless joins, as in [1], [22], [26], and it is the notion of equivalence underlying the normal form decompositions of [9], [10].

We shall deal with weak equivalence, which we hereafter call simply *equivalence*, almost exclusively in this paper, ending with a demonstration of how our ideas carry over to strong equivalence as well. The motivation for so doing is not our belief that strong equivalence is an inferior notion; rather our ideas are more simply expressed when (weak) equivalence is considered. In particular, we may take advantage of the presence of universal instances to regard the value of an expression as a mapping from instances to relations. That is, if I is an instance, $\nu_I(E)$ is the value of expression E when each argument R_i of E is replaced by $\pi_{R_i}(I)$.

Example 3. Consider the expression $E = \pi_{AB}(AB \bowtie BC)$. Here, A, B and C are attributes, and relation schemes are denoted by strings of attributes, i.e., AB stands for $\{A, B\}$. If there is a univeral instance I over attributes A, B and C, in that order, then the value r_{AB} for relation scheme AB is

$$\{ab | \text{for some } c, abc \text{ is in } I\}$$

and the value of r_{BC} for BC is

$$\{bc | \text{for some } a, abc \text{ is in } I\}.$$

The value of $AB \bowtie BC$ is

$$r_{AB} \bowtie r_{BC} = \{abc | \text{for some } a' \text{ and } c', abc' \text{ and } a'bc \text{ are in } I\}.$$

Finally, the value of E is

(*) $$\{ab | \text{for some } a', c' \text{ and } c, abc' \text{ and } a'bc \text{ are in } I\}$$

which is just

$$\{ab | \text{for some } c, abc \text{ is in } I\}$$

as we may take $a = a'$ and $c = c'$ in (1). Thus, E is equivalent to the expression consisting of the single relation scheme AB.

On the other hand, consider strong rather than weak equivalence. Then r_{AB} and r_{BC} can be independently chosen relations. The value of E is

$$\{ab | \text{for some } c, ab \text{ is in } r_{AB} \text{ and } bc \text{ is in } r_{BC}\}$$

which is not necessarily equal to r_{AB}. For example, if $r_{AB} = \{ab\}$ and $r_{BC} = \varnothing$, then the value of E is $\varnothing$, not $\{ab\}$. Note that these values for r_{AB} and r_{BC} cannot come from one instance.

2.8. The effect of data dependencies. Constraints, such as functional dependencies, also affect the requirements for equivalence of expressions. For example, functional dependencies may be applied to instances, and in the presence of a set of functional dependencies we say that $E_1 \equiv E_2$ if $\nu_I(E_1) = \nu_I(E_2)$ for all instances I that satisfy the functional dependencies. Similarly, functional dependencies may apply to relations, and we define E_1 to be strongly equivalent to E_2 in the presence of functional dependencies, if $\nu_\alpha(E_1) = \nu_\alpha(E_2)$ for all assignments α of relations r_i to arguments R_i such that the r_i's satisfy the dependencies.

3. Tableaux. In this section we show how to represent the mappings defined by relational expressions by specialized matrices called "tableaux". Tableaux are similar to the tabular queries of Query-by-Example [27] and the conjunctive queries of Chandra and Merlin [8]. We shall see that for every query in our query language there is a tableau with the same value, but unfortunately, the correspondence is not exact. There are tableaux that do not correspond to any expression over the operators we discuss (or, to our knowledge, over any other set of operators that have appeared in the literature).

3.1. Definition of a tableau. A tableau is a matrix in which the columns correspond to the attributes of the universe in a fixed order. The first row of the matrix is called the *summary* of the tableau. The remaining rows are to be exclusively called *rows.*

The general idea is that a tableau is a shorthand for an explicit set description, such as (*) above, used to define the value of an expression. The summary represents what

appears to the left of the vertical bar, e.g., ab in (*) The rows represent the tuples required to be in I, such as abc' and $a'bc$ in (*)

To simplify later discussion we shall adopt the following conventions regarding tableaux. The symbols appearing in a tableau are chosen from:

(1) Distinguished variables, for which we use a's, possibly with subscripts. These correspond to the symbols to the left of the bar, as a and b in (*).
(2) Nondistinguished variables, for which we generally use b's. These are the other symbols appearing in set formers, such as a' and c' in (*).
(3) Constants, for which we use c's or nonnegative integers.
(4) Blank.

The summary of a tableau may contain only distinguished variables, constants, and blanks. The rows of a tableau may contain variables (distinguished and nondistinguished) and constants. We also require that the same variable not appear in two different columns of a tableau, and that a distinguished variable not appear in a column unless it also appears in the summary of that column.

Let T be a tableau and let S be the set of all symbols appearing in T (i.e., variables and constants). A *valuation* ρ for T associates with each symbol of S a constant, such that if c is a constant in S, then $\rho(c) = c$. We extend ρ to the summary and rows of T as follows. Let w_0 be the summary of T, and $w_1, w_2, \cdots, w_n$ the rows. Then $\rho(w_i)$ is the tuple obtained by substituting $\rho(v)$ for every variable v that appears in w_i.

A tableau defines a mapping from instances to relations on a certain subset of attributes, called the *target relation scheme*, in the following way. If T is a tableau and I an instance, then $T(I)$ is the relation on the attributes whose columns are nonblank in the summary, such that

$$T(I) = \{\rho(w_0) | \text{for some valuation } \rho \text{ we have } \rho(w_i) \text{ in } I \text{ for } 1 \leqq i \leqq n\}.$$

Example 4. Let T be the tableau

A	B	C
a_1	a_2	
a_1	b_1	b_3
b_2	a_2	1
b_2	b_1	b_4

We conventionally show the summary first, with a line below it. We can interpret this tableau as defining the following relation on AB

$$T(I) = \{a_1a_2 | (\exists b_1)(\exists b_2)(\exists b_3)(\exists b_4) \text{ such that } a_1b_1b_3 \text{ is in } I \text{ and } b_2a_21 \text{ is in } I \text{ and } b_2b_1b_4 \text{ is in } I\}$$

where I is any instance. For example, suppose I is the instance $\{111, 222, 121\}$.

Consider the valuation ρ which assigns 1 to all the variables. Under this valuation, the three rows of T each become 111, which is a member of I. Therefore, $\rho(a_1a_2) = 11$ is in $T(I)$.

If ρ assigns 2 to b_1 and a_2, and 1 to the other variables, all rows become 121, so $\rho(a_1a_2) = 12$ is in $T(I)$.

If ρ assigns 2 to a_1, b_1 and b_3, and 1 to the other variables, then $\rho(a_1b_1b_3) = 222$ is in I, $\rho(b_2a_21) = 111$ is in I, and $\rho(b_2b_1b_4) = 121$ is in I, so $\rho(a_1a_2) = 21$ is in $T(I)$.

Finally, if ρ assigns 1 to b_2 and b_4, and 2 to the other variables, then we see that 22 is in $T(I)$. Thus, $T(I) = \{11, 12, 21, 22\}$. □

Conventionally, we also regard $\varnothing$ as a tableau. This tableau represents the function that maps every instance to the empty relation.

Tableaux are closely related to the conjunctive queries of [8]. The significant differences between tableaux and conjunctive queries are that

(1) tableaux permit constants in the summary,
(2) columns of a tableau are associated with attributes, and
(3) tableaux do not permit symbols appearing in two different columns.

Condition (1) is needed to handle the select operator; condition (2) is required that we may talk about dependencies and their effect on equivalence of expressions. Condition (3) is assumed because it enables us to show that even restricted subsets of conjunctive queries have hard optimization problems, and, more importantly, it enables us to isolate a large subset of tableaux for which optimization is relatively easy.

3.2. Equivalence of tableaux. Two tableaux T_1 and T_2 are *equivalent*, written $T_1 \equiv T_2$, if for all I, $T_1(I) = T_2(I)$. We say that T_1 is *contained in* T_2, written $T_1 \subseteq T_2$, if for all I, $T_1(I) \subseteq T_2(I)$. Note that a necessary, but not sufficient, condition for both $T_1 \equiv T_2$ and $T_1 \subseteq T_2$ is that the relations defined by T_1 and T_2 have the same target relation scheme.

As we shall see, the questions of equivalence and containment of tableaux are in the general case hard combinatorial problems. We can, however, state a basic and not unexpected result, namely that consistent renaming of variables does not change the value of a tableau, thus providing many obvious equivalences.

LEMMA 1. *Let T be a tableau and ψ a one-to-one correspondence that maps distinguished variables to distinguished variables, nondistinguished variables to nondistinguished variables, and constants to constants. If we construct a tableau T' from T by simultaneously substituting $\psi(v)$ for every occurrence of symbol v in T, then $T \equiv T'$.*

Proof. This result follows immediately from the definitions. □

3.3. Representation of expressions by tableaux. In this section we show how to construct a tableaux to represent any expression over the operators select, project, and join. The construction proceeds inductively by first building tableaux for the individual operands of an expression, and then combining these tableaux to form tableaux for larger and larger subexpressions, until a tableau for the entire expression is found. The rules for building a tableaux T for an expression E are:

(1) If E is a single relation scheme R, then the tableau T for E has one row and a summary such that:
  (i) If A is an attribute in R, then in the column for A, tableau T has the same distinguished variable in the summary and row.
  (ii) If A is not in R, then its column has a blank in the summary and a nondistinguished variable in the row.

(2a) Suppose E of the form $\sigma_{A=c}(E_1)$, and we have constructed T_1, the tableau for E_1.
  (i) If the summary for T_1 has blank in the column for A, then $T = \varnothing$.
  (ii) If there is a constant $c' \neq c$ in the summary column for A, then $T = \varnothing$. If $c = c'$, then $T = T_1$.
  (iii) If T_1 has a distinguished variable a in the summary column for A, the tableau T for E is constructed by replacing a by c whenever it appears in T_1.

(2b) Suppose E is of the form $\pi_X(E_1)$, and T_1 is the tableau for E_1. The tableau T for E is constructed by replacing nonblank symbols by blanks in the summary of T_1 for those columns whose attributes are not in X. Distinguished variables in those columns become nondistinguished.

(2c) Suppose E is of the form $E_1 \bowtie E_2$ and T_1 and T_2 are the tableaux for E_1 and E_2, respectively. Let S_1 and S_2 be the symbols of T_1 and T_2, respectively. By Lemma 1, we may take S_1 and S_2 to have disjoint sets of nondistinguished variables, but identical distinguished variables in corresponding columns.

(i) If T_1 and T_2 have some column in which their summaries have distinct constants, then $T = \varnothing$.

(ii) If no corresponding positions in the summaries have distinct constants, the rows of the tableau T for E consist of the union of all the rows of T_1 and T_2. The summary of T has in a given column

(a) The constant c if one or both of T_1 and T_2 have c in that column's summary. In this case we also replace any distinguished variable in that column by c.

(b) The distinguished variable a if (a) does not apply, but one or both of T_1 and T_2 have a in that column's summary.

(c) Blank, otherwise.

THEOREM 1. *The rules above construct for any restricted relational expression E a tableau T such that for all instances I, $\nu_I(E) = T(I)$.*

Proof. The proof is an induction on the number of operators in E.

Basis. Rule (1). If there are no operators in E, then E is a single relation scheme R, and rule (1) clearly constructs the appropriate tableau T.

Induction. Rule (2a). $E = \sigma_{A=c}(E_1)$. Let T_1 be the tableau for E_1.

(i) If the summary for T_1 has blank in the column for A, then the expression E has no meaning and $\varnothing$ is the correct tableau for E.

(ii) If there is a constant $c' \neq c$ in the summary column for A, then for any I, $\nu_1(E_1)$ has only tuples with c' in the component for A, and $\nu_I(E)$ is empty. Again, $\varnothing$ is the correct tableau for E. If $c = c'$, then T_1 is the correct tableau for E.

(iii) If T_1 has a distinguished variable a in the summary column for A, and we construct T for E by replacing a by c whenever it appears in T_1, then we claim that for all I, $T(I) = \sigma_{A=c}(T_1(I))$. In proof, suppose ρ is a map from the symbols of T_1 to a set of constants C. Let $w_0, w_1, \cdots, w_n$ be the summary and rows of T_1, and let $w_0', w_1', \cdots, w_n'$ be the same for T. That is, w_i' is w_i with a replaced by c if a appears in w_i. Then,

$$\begin{aligned} T(I) &= \{\rho(w_0') | \rho(w_i') \text{ is in } I \text{ for } 1 \leqq i \leqq n\} \\ &= \{\rho(w_0) | \rho(a) = c \text{ and } \rho(w_i) \text{ is in } I \text{ for } 1 \leqq i \leqq n\} \\ &= \sigma_{A=c}(\{\rho(w_0) | \rho(w_i) \text{ is in } I \text{ for } I \leqq i \leqq n\}) \\ &= \sigma_{A=c}(T_1(I)). \end{aligned}$$

The third line above follows from the fact that w_0 is known to have a in its column for A.

Rule (2b). $E = \pi_X(E_1)$. A proof of the correctness of this case is straightforward and is omitted.

Rule (2c). $E = E_1 \quad E_2$.

(i) If T_1 and T_2 have some column in which their summaries have distinct constants, then $V(E)$ maps all instances to $\varnothing$, so $\varnothing$ is the correct tableau for E.

(ii) If no corresponding positions in the summaries have distinct constants, we claim that $T(I) = T_1(I) \bowtie T_2(I)$ for all I. Let w_0 be the summary of T. Let x_j, $0 \leqq i \leqq n_1$, and y_i, $0 \leqq i \leqq n_2$, be the summaries and rows of T_1 and T_2, respectively. Then

$$T_1(I) = \{\rho_1(x_0) | \rho_1(x_i) \text{ is in } I \text{ for } 1 \leqq i \leqq n_1\},$$

$$T_2(I) = \{\rho_2(y_0) | \rho_2(y_i) \text{ is in } I \text{ for } I \leqq i \leqq n_2\},$$

$T_1(I) \bowtie T_2(I) = \{\rho(w_0) |$ for some ρ_1 and ρ_2, ρ agrees with ρ_1 and/or ρ_2, respectively, on the attributes with nonblank symbols in x_0 and y_0, respectively, $\rho_1(x_i)$ is in I for $1 \leqq i \leqq n_1$, and $\rho_2(y_i)$ is in I for $1 \leqq i \leqq n_2\}$.

As S_1 and S_2 have disjoint sets of nondistinguished variables, we may extend ρ to agree with ρ_1 and ρ_2 on all symbols present in T. Therefore

$$T_1(I) \bowtie T_2(I) = \{\rho(w_0) | \rho(x_i) \text{ is in } I \text{ for } 1 \leqq i \leqq n_1 \text{ and } \rho(y_i) \text{ is in } I \text{ for } I \leqq i \leqq n_2\}.$$ □

Example 5. Let A, B and C be the attributes, in that order, and suppose we have the expression $\pi_{AC}(\sigma_{B=0}(AB \bowtie BC))$. By Rule (1), the tableaux for AB and BC are

A	B	C
a_1	a_2	
a_1	a_2	b_1

and

A	B	C
	a_2	a_3
b_2	a_2	a_3

By Rule (2c), the tableau for $AB \bowtie BC$ is

A	B	C
a_1	a_2	a_3
a_1	a_2	b_1
b_2	a_2	a_3

By Rule (2a), the tableau for $\sigma_{B=0}(AB \bowtie BC)$ is

A	B	C
a_1	0	a_3
a_1	0	b_1
b_2	0	a_3

Finally, by Rule (2b), the tableau for $\pi_{AC}(\sigma_{B=0}(AB \bowtie BC))$ is

A	B	C
a_1		a_3
a_1	0	b_1
b_2	0	a_3

□

It is interesting to note that Chandra and Merlin [8] prove an analogue of Theorem 1 and also its converse, using select, project and join operations that are suitably generalized to take advantage of the fact that columns are not pinned down to particular attributes, and also an operator called restriction, that in effect identifies two distinguished variables of the same relation. However, in our model the converse to Theorem 1 is false. That is, there are tableaux that come from no expression, as the following example shows.

Example 6. The tableau

a_1	a_2
a_1	b_2
b_1	a_2
b_1	b_2

cannot be derived from any restricted relational expression. If there is such an expression, suppose that the last two rows come from the first two relations joined. The expression resulting from this join must later be joined with a relation from which the first row, a_1b_2, is derived. Since b_2 appears in rows 1 and 3, b_2 must have been distinguished at this later time, else the symbols in these positions could not be identified with one another. Since a_2 is currently distinguished, however, it must have been so when the last join was performed, and symbols b_2 and a_2 would not be distinct. A similar contradiction is obtained no matter which two rows we assume are grouped first.

In fact, even had we introduced a restriction operator, we could not produce the above tableau. In proof, note that if a tableau has a symbol appearing in two columns, the operations on tableaux corresponding to select, project and join preserve that property. Since the above tableau has no symbol in both columns, we know that restriction could be of no help in forming it. □

We know of no natural set of operators that characterizes tableaux exactly.

The construction rules above can also be used to define the operations select, project and join on tableaux. The result of applying any one of these operations to tableaux (not necessarily tableaux derived from expressions) is defined to be the tableau described in the rule for that operation.

4. Testing equivalence of tableaux. In this section we shall give a method for testing the equivalence of tableaux, thus providing an algorithm for testing the equivalence of expressions.

4.1. Homomorphisms. Chandra and Merlin [8] give a necessary and sufficient condition for the equivalence of conjunctive queries in terms of "homomorphisms," which are symbol-symbol mappings with certain properties. We shall prove the analogous result here for tableaux. We shall then prove a dual formulation of the equivalence test of [8] in terms of row-row mappings called "containment mappings."

Let T_1 and T_2 be two tableaux with sets of symbols S_1 and S_2. A *homomorphism* is a mapping $\psi : S_1 \to S_2$ such that:

(i) If c is a constant, then $\psi(c) = c$.

(ii) If a is distinguished, then $\psi(a)$ either is distinguished or is the constant appearing in the corresponding column of the summary of T_2.

(iii) If w is any row of T_1, then $\psi(w)$ is a row of T_2.

Then, intuitively, any time that we can map the rows of T_2 into elements of an instance I, the homomorphism ψ gives us a map from rows of T_1 into I as well. Thus, $T_2(I) \subseteq T_1(I)$ for all I, so $T_2 \subseteq T_1$.

The converse holds as well. If $T_2 \subseteq T_1$, then we can make the rows of T_2 be an instance I of the universe, by treating all symbols of S_2 as distinct constants. The fact that $T_2 \subseteq T_1$ implies that $T_2(I) \subseteq T_1(I)$. The fact that the summary of T_2, with blanks deleted, is in $T_2(I)$, and hence in T_1(I), implies that the homomorphism $\psi : S_1 \to S_2$ exists. We may formalize the above as follows.

THEOREM 2. *Let T_1 and T_2 be two tableaux with sets of symbols S_1 and S_2. $T_2 \subseteq T_1$ if and only if they have the same target relation scheme, and there is a homomorphism $\psi : S_1 \to S_2$.*

Proof [8] (*If*). Let I be an instance, and let $\rho : S_2 \to C$ be a valuation, where C is a set of constants, such that for each row w of T_2, $\rho(w)$ is an element of I. Then $\rho \cdot \psi : S_1 \to C$ is a valuation that sends each row of T_1 to an element of I, by condition (iii). By conditions (i) and (ii), if s_1 and s_2 are the summaries of T_1 and T_2, respectively, with blanks deleted, then $\rho(s_2) = \rho(\psi(s_1))$. Thus, any tuple in $T_2(I)$ is in $T_1(I)$, so $T_2 \subseteq T_1$.

(*Only if*). Let ρ be a one-to-one correspondence between the symbols of T_2 and some set of constants, and let I be the instance consisting of all the elements $\rho(w)$, for w a row of T_2. Then $\rho(s_2)$ is in $T_2(I)$, and since $T_2 \subseteq T_1$, it is also in $T_1(I)$. Thus, there is a homomorphism $\psi : S_1 \to S_2$ satisfying (i)–(iii) by the definition of the application of a tableau to an instance and the fact that ρ is one-to-one. □

4.2. Containment mappings. A containment mapping is a mapping from the rows of one tableau to another that preserves distinguished variables and constants and does not map any symbol to two different symbols. Formally, let T_1 and T_2 be tableaux, and let θ be a mapping from the rows of T_1 to the rows of T_2. We say θ is a *containment mapping* if:

(a) For each row i of T_1, if row i has a distinguished variable in some column A, then row $\theta(i)$ of T_2 has a distinguished variable or constant in column A.

(b) If row i of T_1 has a constant c in column A, then row $\theta(i)$ has c in column A.

(c) If rows i and j of T_1 have the same nondistinguished variable in column A, then rows $\theta(i)$ and $\theta(j)$ have the same symbol in that column. That symbol could be constant, distinguished, or nondistinguished. Also note that $\theta(i) = \theta(j)$ is possible.

We may prove the following analogue to Theorem 2.

THEOREM 3. *$T_2 \subseteq T_1$ if and only if they define the same target relation and there is a containment mapping θ from T_1 to T_2.*

Proof (*If*). Let $\psi : S_1 \to S_2$ be a symbol-symbol mapping such that if symbol d appears in column A of row r of T_1, and symbol d' appears in column A of row $\theta(r)$ of T_2, then $\psi(d) = d'$. The map ψ is consistent by condition (c). Conditions (i)–(iii) for ψ are immediate. That is, (a) implies (ii), (b) implies (i), and (iii) is implied by the definition of ψ from θ. Thus, ψ is a homormorphism, and by Theorem 2, $T_2 \subseteq T_1$.

(*Only if*). By Theorem 2, there is a homomorphism $\psi : S_1 \to S_2$ satisfying (i)–(iii). The existence of a map from the rows of T_1 to the rows of T_2 satisfying (c) follows from (iii); (i) and (ii) imply (a) and (b). □

As a containment mapping on rows induces a homomorphism satisfying (i)–(iii), we shall sometimes fail to distinguish a containment mapping from its corresponding homomorphism.

COROLLARY 1. *$T_1 \equiv T_2$ if and only if T_1 and T_2 have identical summaries up to renaming of distinguished variables, and containment mappings exist in both directions. In this case the possibility that row $\theta(i)$ in condition* (a) *has a constant can be ignored, since a constant cannot map back to a distinguished variable.*

Example 7. The expression $\pi_{AB}(AB \bowtie BC)$ of Example 3 has tableau

	A	B	C
	a_1	a_2	
$T_1 = w_1$	a_1	a_2	b_1
w_2	b_2	a_2	b_3

while the expression AB over set of attributes A, B and C has tableau

$T_2 =$

	A	B	C
	a_1	a_2	
w_3	a_1	a_2	b_1

.

In one direction, the map that sends both w_1 and w_2 to w_3 is a containment mapping. The induced homomorphism is:

in T_1	in T_2
a_1	a_1
a_2	a_2
b_1	b_1
b_2	a_1
b_3	b_1

In the opposite direction, we may map w_3 to w_1, showing the containment in the opposite direction as well. Thus AB and $\pi_{AB}(AB \bowtie BC)$ are equivalent.

For another example, let $E_1 = AB \bowtie AC \bowtie BC$ and $E_2 = ABC \bowtie \sigma_{C=0}(BC)$. The tableaux for E_1 and E_2 are, respectively,

	A	B	C
	a_1	a_2	a_3
$T_1 = w_1$	a_1	a_2	b_1
w_2	a_1	b_2	a_3
w_3	b_3	a_2	a_3

	A	B	C
	a_1	a_2	0
$T_2 = w_4$	a_1	a_2	0
w_5	b_1	a_2	0

.

Then $T_2 \subseteq T_1$, since we may produce a containment mapping by sending w_1, w_2 and w_3 to w_4. We may alternatively map w_3 to w_5 if we like. However, in the opposite direction there is no containment mapping, since the constant 0 cannot map to a variable. Thus $T_1 \not\subseteq T_2$. To prove this we may make an instance I from the rows of T_1 by assigning, say, $1, 2, \cdots, 6$ to a_1, a_2, a_3, b_1, b_2 and b_3. Then $T_1(I)$ contains 123, but $T_2(I)$ does not. □

An additional corollary to Theorem 3 gives a simple row elimination rule for tableaux.

COROLLARY 2. *Let T be a tableau, w some row of T, and suppose there is some other row x of T such that in whatever column w and x disagree, w has a nondistinguished variable that appears nowhere else in T. Then the tableau T', obtained by deleting row w from T, is equivalent to T.*

Proof. We may map each row of T' to itself in T, and we may map each row of T other than w to itself, while mapping w to x. □

Example 8. In the first part of Example 7, row w_2 may be eliminated by w_1, which immediately transforms T_1 into T_2 and proves their equivalence. □

We state without proof two additional results for tableaux. There are analogous results for conjunctive queries [8].

THEOREM 4. *If T_1 and T_2 are equivalent tableaux, and neither is equivalent to a tableau with fewer rows, then there is a one-to-one correspondence of rows of T_1 to rows of T_2 that is a containment map in both directions.*

THEOREM 5. *Given any tableau T we can create a minimum row tableau equivalent to T by deleting some rows of T.*

Theorems 4 and 5 imply that for every tableau T there is a minimum row tableau equivalent to T that is unique up to renaming of symbols and reordering of rows; moreover, this minimum row tableau can be found by removing some of the rows of T. In § 5 we shall see that it is, nevertheless, a computationally difficult task to determine which rows of a tableau are redundant.

4.3. The effect of functional dependencies. When functional dependencies are present, we can use them to transform tableaux to equivalent forms. This can be done in the following way. Suppose X is a set of attributes, A is an attribute, and $X \to A$. Suppose also that two rows i and j of T have identical symbols in all columns corresponding to attributes of X. Let T' be constructed from T as follows.

(a) If rows i and j have two distinct constants in the column corresponding to A, then T' is $\varnothing$.

(b) Otherwise, make the symbols found in row i and row j of column A identical. If one of them is a constant then the resulting symbol is the same constant; if both of them are variables and one is distinguished, so is the resulting symbol.

LEMMA 2. *If T' is obtained from T as described above, then $T(I) = T'(I)$ for every instance I that satisfies the functional dependency $X \to A$.*

Proof. Let d_1 and d_2 be the symbols identified, and let d_3 be the symbol that replaces them in T'. Let S and S' be the sets of symbols of T and T' respectively. Suppose that I is any instance that satisfies $X \to A$, and $\rho : S \to C$ is a valuation under which each row of T becomes a member of I. Since I satisfies $X \to A$, we must have $\rho(d_1) = \rho(d_2)$. Define an assignment $\rho' : S' \to C$ as follows

$$\rho'(d) = \rho(d) \quad \text{if} \quad d \neq d_3, \quad \text{and} \quad \rho'(d_3) = \rho(d_1).$$

The application of ρ and ρ' to T and T' respectively produces identical results, and therefore $T(I) \subseteq T'(I)$.

The converse, that $T'(I) \subseteq T(I)$, is proved in a similar way. □

Example 9. Consider the expression $\pi_{AC}(AB \bowtie BC) \bowtie (AB \bowtie AD)$ whose syntax tree is shown in Fig. 2.

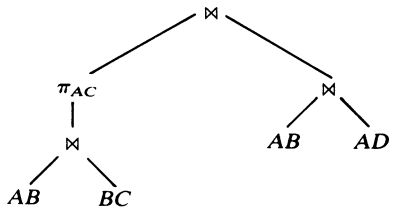

FIG. 2. *Syntax tree for expression.*

The tableau for this expression is

A	B	C	D
a_1	a_2	a_3	a_4
a_1	b_1	b_2	b_3
b_4	b_1	a_3	b_5
a_1	a_2	b_6	b_7
a_1	b_8	b_9	a_4

Suppose the functional dependencies $B \to A$ and $A \to C$ hold. Then $B \to A$ implies that $a_1 = b_4$, and then $A \to C$ implies that all of b_2, a_3, b_6 and b_9 are the same. Therefore the above tableau is equivalent to:

A	B	C	D
a_1	a_2	a_3	a_4
a_1	b_1	a_3	b_3
a_1	b_1	a_3	b_5
a_1	a_2	a_3	b_7
a_1	b_8	a_3	a_4

By Corollary 2 to Theorem 3, the first row may be eliminated in favor of the second row (or vice-versa), and then the remaining of these may be eliminated in favor of the third row, leaving

A	B	C	D
a_1	a_2	a_3	a_4
a_1	a_2	a_3	b_7
a_1	b_8	a_3	a_4

which implies that the given expression is equivalent to $ABC \bowtie ACD$ in the presence of the dependencies $B \to A \to C$. □

Suppose that T is a tableau and F is a given set of functional dependencies. Let i and j be two rows of T, and let X be the set of all the attributes whose corresponding columns have identical symbols in row i and row j. For every column in X^*, we can equate the symbols that appear in this column in row i and row j wherever they appear in T. This process can be applied recursively until no more symbols can be equated. The result is a tableau T' that is equivalent to T for every instance in which F holds, by Lemma 2. It is easy to show that T' is unique for T up to renaming of variables, since the above transformation on tableaux is a "Finite Church–Rosser System" [3]. Informally, if two symbols can be equated, they will always be equatable, no matter what other symbols are equated.

If no symbols of T may be equated because of a set of functional dependencies F, we say *T satisfies F*. The result T' of equating symbols of any tableau T according to the above rules, until no more can be equated is called the *limit of T with respect to F*. By using the algorithm of [5], [6] to compute X^* for sets of attributes X, we can construct the limit of T in time proportional to the square of the input size (the space needed to write down F and T). The algorithm is essentially that given in [1]. In the next theorem we show that in the presence of functional dependencies there is a weaker necessary and sufficient condition for inclusion or equivalence among tableaux.

THEOREM 6. *Let T_1 and T_2 be tableaux with limits T_1' and T_2' with respect to a set of functional dependencies F. Then $T_1(I) \supseteq T_2(I)$ for all instances I satisfying F if and only if $T_1' \supseteq T_2'$.*

Proof. By Lemma 2, $T_1(I) = T_1'(I)$ for all I satisfying F, and similarly for T_2 and T_2'. Thus the "if" portion is immediate. The converse is similar to the "only if" portion of Theorem 2. Here, we make T_2' into an instance I by assigning distinct constants to all its symbols. As T_2' satisfies F, I satisfies F. If $T_1(I) \supseteq T_2(I)$, then $T_1'(I) \supseteq T_2'(I)$. The existence of a homomorphism ψ from the symbols of T_1' to those of T_2' follows as in that theorem. Thus by Theorem 2, $T_1' \supseteq T_2'$. □

COROLLARY. *$T_1(I) = T_2(I)$ for all instances I satisfying F if and only if $T_1' \equiv T_2'$.*

Example 10. Let us continue Example 9, where the dependencies were $B \to A$ and $A \to C$. Consider the expression $AB \bowtie BC \bowtie AD$, whose tableau is

A	B	C	D
a_1	a_2	a_3	a_4
a_1	a_2	b_1	b_2
b_3	a_2	a_3	b_4
a_1	b_5	b_6	a_4

The limit of this tableau is

A	B	C	D
a_1	a_2	a_3	a_4
a_1	a_2	a_3	b_2
a_1	a_2	a_3	b_4
a_1	b_5	a_3	a_4

which is equivalent to the limiting tableau of Example 9, since the first row may be eliminated by Corollary 2 to Theorem 3. Thus the expression of Fig. 3 is equivalent to $AB \bowtie BC \bowtie AD$ if the dependencies $B \to A$ and $A \to C$ are given. Note that these expressions are not equivalent in general. □

5. NP-Completeness results concerning tableau equivalence. The obvious way to test the equivalence of two tableaux is to consider all possible containment mappings in each direction. Since the number of mappings from n_1 rows to n_2 rows is $n_2^{n_1}$, this procedure takes exponential time. One might therefore be interested in finding a procedure that takes less time. Using recent developments in complexity theory, however, we can prove that a substantially better algorithm is not likely to exist.

We assume the reader is familiar with the notion of an NP-complete problem. This class of problems was first considered in [12], [18]. There is strong evidence that these problems are intractable in general, that is, there is no algorithm for any of these problems which, on every input, will take less than exponential time. References [2], [16] present the methodology and theory behind NP-completeness results, as well as enumerating many of the known NP-complete problems.

In this section we show that the equivalence and containment problems for tableaux are NP-complete even in the following special cases:

(1) The tableaux come from expressions that have no select operators, but there is a set of functional dependencies that must be satisified.

(2) The tableaux come from expressions (including select operators), but no dependencies need be satisfied.

(3) There are no constants in the tableaux, nor are there dependencies, but the tableaux need not come from expressions.

Under the same conditions, the problem of determining whether $T_1 \subseteq T_2$ for two tableaux T_1 and T_2 is also NP-complete. Moreover, even if T_1 is a tableau with the same summary as T_2, and the rows of T_1 are a subset of those of T_2, it is NP-complete to determine whether $T_1 \equiv T_2$. This implies that minimizing the rows of a tableau is also very likely an exponential process in the worst case. Our NP-completeness results

strengthen those in [8] since our restricted relational expressions are a subset of the class of conjunctive queries.

5.1. The satisfiability problem. All the results use almost the same reduction from the 3-satisfiability problem, shown NP-complete in [12]; see also [2], [16]. Let $F = F_1F_2 \cdots F_q$ be a Boolean expression in conjunctive normal form, where the F_i's are clauses of three literals[1] each, and $x_1, x_2, \cdots, x_n$ are all the variables appearing in this expression. We construct two tableaux T_1 and T_2, each with $n+q$ columns, in the following way. T_1 has one row for each clause F_i. Let w_i be the row that corresponds to F_i. Let x_{i_1}, x_{i_2} and x_{i_3} be the variables that appear in F_i, either complemented or uncomplemented. Row w_i has the distinguished variable a_i in the ith column and the nondistinguished variables x_{i_1}, x_{i_2} and x_{i_3} in columns $q+i_1$, $q+i_2$ and $q+i_3$, respectively. The rest of the columns of w_i contain nondistinguished variables that appear nowhere else. The summary of T_1 has a_i in the ith column, $1 \leqq i \leqq q$, and blank in the other columns.

T_2 has seven rows for each row of T_1. Let w_i be a row of T_1. Each of the seven rows of T_2 that correspond to w_i represents some truth assignment to the variables of F_i under which F_i is true. Such a row has the distinguished variable a_i in the ith column and one of the seven lists of constants c_{i_1}, c_{i_2} and c_{i_3} in columns $q+i_1$, $q+i_2$ and $q+i_3$, respectively, such that each c_{i_j} is zero or one, and the assignment of the set of values c_{i_j} to $x_{i_j}(1 \leqq j \leqq 3)$ results in F_i being true. The rest of the columns contain distinct nondistinguished variables. The summary of T_2 is the same as that of T_1.

Example 11. Consider the Boolean expression

$$(x_1 + \bar{x}_2 + x_3)(\bar{x}_3 + x_4 + x_5).$$

Then $F_1 = (x + \bar{x}_2 + x_3)$ and $F_2 = (\bar{x}_3 + x_4 + x_5)$; q is 2 and n is 5. T_1 is:

F_1	F_2	x_1	x_2	x_3	x_4	x_5
a_1	a_2					
a_1	b_1	x_1	x_2	x_3	b_2	b_3
b_4	a_2	b_5	b_6	x_3	x_4	x_5

The seven rows of T_2 that correspond to the first row of T_1 are

$$(a_1, b_7, 1, 1, 1, b_8, b_9)$$
$$(a_1, b_{10}, 1, 1, 0, b_{11}, b_{12})$$
$$(a_1, b_{13}, 1, 0, 1, b_{14}, b_{15})$$
$$(a_1, b_{16}, 1, 0, 0, b_{17}, b_{18})$$
$$(a_1, b_{19}, 0, 1, 1, b_{20}, b_{21})$$
$$(a_1, b_{22}, 0, 0, 1, b_{23}, b_{24})$$
$$(a_1, b_{25}, 0, 0, 0, b_{26}, b_{27}).$$

[1] A *literal* is a variable or negated variable.

The rows of T_2 that correspond to the second row of T_1 are

$$(b_{28}, a_2, b_{29}, b_{30}, 1, 1, 1)$$
$$(b_{31}, a_2, b_{32}, b_{33}, 1, 1, 0)$$
$$(b_{34}, a_2, b_{35}, b_{36}, 1, 0, 1)$$
$$(b_{37}, a_2, b_{38}, b_{39}, 0, 1, 1)$$
$$(b_{40}, a_2, b_{41}, b_{42}, 0, 1, 0)$$
$$(b_{43}, a_2, b_{44}, b_{45}, 0, 0, 1)$$
$$(b_{46}, a_2, b_{47}, b_{48}, 0, 0, 0).$$

Note that the first seven rows do not include the combination 0, 1, 0, because if we assign 0 to x_1 and x_3 and 1 to x_2, then $F_1 = (x_1 + \bar{x}_2 + x)$ gets the truth value 0. Similarly, the last seven rows do not contain the combination 1, 0, 0. □

5.2. A class of tableaux that come from expressions. It happens that T_1 and T_2 are both obtainable from expressions by the construction preceding Theorem 1. These observations are special cases of a more general result, which we state as the next lemma. A *repeated symbol* in a particular column of a tableau is either

(1) a distinguished variable,
(2) a constant appearing in that column of the summary,
(3) a nondistinguished variable appearing in two or more rows.

Notice that a repeated symbol might appear in only one row if it is a distinguished variable or a constant appearing in the summary.

LEMMA 3. *If T is a tableau with at most one repeated symbol in any column, and such that any symbol appearing in the summary appears in at least one row in the same column, then there is a relational expression E, such that Theorem 1 applied to E yields T.*

Proof. For each row i of T, let R_i be the relation scheme consisting of the attributes in whose columns row i has a repeating symbol or other constant. Construct expression E_i by applying $\sigma_{A=c}$ to R_i for all attributes A whose column in row i has a constant c that does not appear in the same column of the summary. The tableau for E_i is a row like row i, but with distinguished variables in place of all repeated symbols, and with a summary containing the distinguished variable in exactly those columns in which row i has a repeated symbol.

Next, join all the E_i's. The result is an expression with a tableau like T, but with distinguished variables for all repeated symbols. Lastly, apply $\sigma_{A=c}$ for all A whose column has in the summary a constant c, and project onto those attributes such that the summary of T has a nonblank. The result is an expression with tableau T. □

COROLLARY. *T_1 and T_2 above come from expressions.*

Proof. T_1 has only the a_i's and (possibly) the x_i's as repeated symbols; T_2 has only the a_i's as repeated symbols.

Example 12. Consider T_1 of Example 11 and suppose the columns correspond to attributes $A_1, A_2, \cdots, A_7$. The repeated symbols are a_1, a_2, and x_3. The relation schemes for the two rows are $R_1 = A_1A_5$ and $R_2 = A_2A_5$. The expression corresponding to T_1 is $\pi_{A_1A_2}(A_1A_5 \bowtie A_2A_5)$. □

5.3. NP-Completeness results for expressions.

LEMMA 4. *Let T_1 and T_2 be constructed from a Boolean expression F as above. Then $T_1 \supseteq T_2$ if and only if F is satisfiable.*

Proof (*If*). Given an assignment that makes F true, we may construct a homomorphism ψ from the symbols of T_1 to those of T_2 as follows.

$\psi(a_i) = a_i$,

$\psi(x_i) = 0$ or 1 depending on the value assigned to x_i to make F true.

We may then map each row w of T_1 to that one of the seven corresponding rows that is $\psi(w)$ when we extend ψ to the rest of the nondistinguished variables. Since each nondistinguished variable of T_1 except for the x_i's appears only once, we can always extend ψ in this manner. Thus $T_1 \supseteq T_2$ by Theorem 2.

(*Only If*). Suppose there is a containment mapping of T_1 to T_2. Because of the a_i's, each row of T_1 must be mapped to one of the seven corresponding rows of T_2. Each x_i is mapped to either 0 or 1 consistently. The values chosen for the x_i's satisfy F, because the combinations of values making clauses false are not available as rows of T_2. Thus F is satisfiable. □

THEOREM 7. *Let U_1 and U_2 be two tableaux that are derived from restricted relational expressions. The following problems are NP-complete*:

(1) *Does $U_1 \supseteq U_2$?*

(2) *Is $U_1 \equiv U_2$?*

(3) *Let U_2 be a tableau that is obtained by deleting some of the rows of U_1. Is $U_1 \equiv U_2$?*

Proof. All these problems are in NP, because all we have to do is to guess a containment mapping and check whether it satisfies all the required conditions. Part (1) is immediate from Lemma 4.

For part (2), let $U_1 = T_1 \bowtie T_2$ and $U_2 = T_2$, where T_1 and T_2 are as above. Recall that the join is defined for tableaux by Theorem 1. Also note that U_1 is obtainable from an expression if T_1 and T_2 are. Since T_1 and T_2 define mappings whose values are relations with the same target relation scheme, the join is really intersection. Thus for any I, $U_1(I) = T_1(I) \cap T_2(I)$, and $U_1 \equiv U_2$ if and only if $T_1 \supseteq T_2$. Thus, equivalence is NP-complete by Lemma 4.

For part (3), simply observe that the rows of U_2 constructed in part (2) are a subset of the rows of U_1 in that part. □

Parts (1) and (2) of Theorem 7 say that the problem of testing equivalence or containment of expressions is almost certainly an intractable one, that is, no general algorithms of less than exponential complexity exist. Part (3) says that the problem of eliminating redundant rows of the tableau derived from one of these expressions is also likely to be intractable.

5.4. NP-Completeness results for tableaux. We should note the critical role played by constants in the proof of Lemma 4 and Theorem 7. However, if we are willing to relax our constraint that the tableaux come from expressions, then constants are not needed.

THEOREM 8. *The problems of Theorem 7 are NP-complete for general tableaux that have no constants.*

Proof. In T_2 defined previously, in each column replace 0 by a nondistinguished variable and 1 by another nondistinguished variable. The proof is then identical to Theorem 7. Note that T_2 does not in general come from any relational expression. □

5.5. NP-Completeness results with functional dependencies. In the presence of functional dependencies, we can prove similar results about tableaux that have only

variables and correspond to expressions with operations project and join only. The key idea is to use tableaux with $q+2n$ columns as follows. The first tableau $\hat{T}_1$ is simply obtained from T_1 by adding another n columns that contain only distinct nondistinguished variables.

To generate the second tableau $\hat{T}_2$, we modify the last n columns of T_2 as follows. First we replace in every column each occurrence of the constant 1 by the same nondistinguished variable, and each occurrence of the constant 0 is replaced by a distinct (for that occurrence) nondistinguished variable. The resulting columns are the $(q+1)$st, $\cdots$, $(q+n)$th columns of $\hat{T}_2$. Columns $q+n+1$ through $q+2n$ of $\hat{T}_2$ are obtained by a similar modification on columns $q+1$ through $q+n$ of T_2; each occurrence of the constant 0 in a particular column is replaced by the same nondistinguished variable, and each occurrence of the constant 1 is replaced by a distinct nondistinguished variable.

Both $\hat{T}_1$ and $\hat{T}_2$ correspond to expressions by Lemma 3. Let A_i be the attribute of the ith column. Suppose that we consider only instances in which the functional dependencies $A_{i+n} \to A_i$ $(q+1 \leqq i \leqq q+n)$ hold. Using these dependencies we can equate all the distinct variables, in the ith column $(q+1 \leqq i \leqq q+n)$, that stand for the truth value 0. Notice that each column between $q+1$ and $q+n$ already has a single symbol representing truth value 1. Therefore, $\hat{T}_1(1) \supseteq \hat{T}_2(I)$, for all instances I satisfying the dependencies, if and only if the Boolean expression F is satisfiable.

As a result of this reduction, we may conclude the following.

THEOREM 9. *Given a set of functional dependencies and two tableaux U_1 and U_2 that come from relational expressions with no select operations (and hence U_1 and U_2 have no constants), it is NP-complete whether, for all instances I satisfying the functional dependencies,*

(1) $U_1(I) \supseteq U_2(I)$

(2) $U_1(I) = U_2(I)$

(3) $U_1(I) = U_2(I)$ *given that the rows of U_2 are a subset of the rows of U_1.*

Proof. Let T_1' and T_2' be the limits of $\hat{T}_1$ and $\hat{T}_2$ above with respect to the functional dependencies given above. Then $T_1' = \hat{T}_1$, and in each of columns $q+1$ through $q+n$ of T_2', there is one nondistinguished variable where T_2, defined previously, has 0, and another where T_2 has 1. Other than this, the first $q+n$ columns of T_2' are the same as T_2. As T_1' has distinct nondistinguished variables in all positions of its last n columns, it follows as in Lemma 4 that there is a containment mapping from T_1' to T_2' if and only if the Boolean expression F is satisfiable. By Theorem 6, $T_1' \supseteq T_2'$ if and only if for all I satisfying the dependencies, $\hat{T}_1(I) \supseteq \hat{T}_2(I)$. Thus F is satisfiable if and only if for all instances I satisfying the dependencies, $\hat{T}_1(I) \supseteq \hat{T}_2(I)$. Parts (2) and (3) follow as in Theorem 7. □

6. A polynomial-time equivalence algorithm for a subclass of tableaux. In this section we define "simple tableaux," a large subclass of tableaux for which we can find a polynomial-time algorithm to decide equivalence.

6.1. Simple tableaux. A tableau is *simple* if in any column with a repeated nondistinguished variable there is no other symbol that appears in more than one row. It is not easy to produce an expression with a nonsimple tableau. The expression $\pi_{AC}(AB \bowtie BC) \bowtie (AB \bowtie BD)$ is in a sense a minimal expression that gives rise to a nonsimple tableau. The tableau is shown in Fig. 3. The rows in the column for B have repeated nondistinguished and distinguished variables.

A	B	C	D
a_1	a_2	a_3	a_4
a_1	b_1	b_2	b_3
b_4	b_1	a_3	b_5
a_1	a_2	b_6	b_7
b_8	a_2	b_9	a_4

FIG. 3. *A nonsimple tableau.*

Note that some simple tableaux do not come from expressions.

Intuitively, the algorithm for equivalence of simple tableaux works as follows. Suppose first that no column has any repeated nondistinguished variables. When we are dealing with equivalence, rather than containment, we can rule out containment mappings in which a distinguished variable maps to a constant. Therefore, to check for the existence of containment mappings in the situation where no nondistinguished variable repeats, we have only to examine each row r to see whether there is another row r' in the other tableau such that r' has a distinguished variable or identical constant wherever r has a distinguished variable or constant.

However, simple tableaux admit repeated nondistinguished variables in a column, provided there is not also another repeated symbol of any sort appearing in two rows of that column. Let T_1 and T_2 be equivalent simple tableaux and A a column of T_1 with repeated nondistinguished variable b_1. As T_1 and T_2 are equivalent, there is a containment mapping θ_1 from T_1 to T_2, and another containment mapping θ_2 from T_2 to T_1. It is easy to check that the composition of containment mappings is a containment mapping, so we may consider the containment mapping $\theta_2 \cdot \theta_1$ from T_1 to itself, as suggested in Fig. 4.

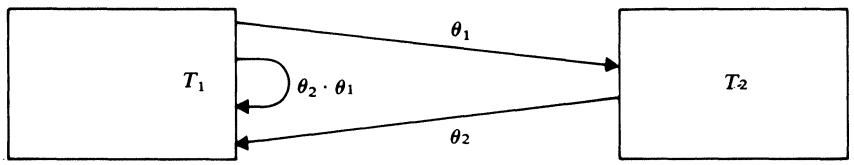

FIG. 4. *Composition of containment mappings.*

Now let us look at the set of rows S of T_1 that have b_1 in column A. There are two cases:

(a) $\theta_2 \cdot \theta_1$ maps rows in S to two or more rows of T_1.

(b) $\theta_2 \cdot \theta_1$ maps all rows in S to a single row r.

In case (b) we can eliminate all rows in S (except r if it is in S) from T_1, and the result will be a tableau equivalent to T_1. In case (a) we know that $\theta_2 \cdot \theta_1(w)$ is in S for all w in S, because by the hypothesis that T_1 is simple, no pair of rows other than those in S have the same symbol in the column for A. Moreover, θ_1 maps S to at least two rows, and these rows must have the same nondistinguished variable in column A. For if they had a distinguished variable or constant, θ_2 could not map them to rows in S. Thus in case (a) there is a repeated nondistinguished variable b_2 in column A of T_2.

Our algorithm works as follows. We search for a column A in which one tableau T_1 has a repeated nondistinguished variable in some set of rows S. If there exists a

containment mapping $\theta_2 \cdot \theta_1$ from T_1 to itself that maps all of S to one row r, then we eliminate S, and perhaps some other rows, to be determined later, in favor of r. If no such mapping exists, then only case (a) can apply, if $T_1 \equiv T_2$. Then θ_1 and θ_2 must map rows with b_1 to rows with b_2, and vice-versa. In this case we may "promote" b_1 and b_2 by treating them as constants. Ultimately, we eliminate all repeated nondistinguished variables, either by row elimination or by promotion. The resulting tableaux meet our earlier requirements for an efficient equivalence test, since they have no repeated nondistinguished variables. We now proceed to formalize the above argument.

6.2. Row covering. We say that row x of a tableau *covers* row w if the following hold.

(a) w and x have the same number of columns.

(b) If w has a distinguished variable in a given column, so does x. If w has a constant in a given column, then x has the same constant in this column.

We say x *covers* a set of rows S if x covers every row in S.

Example 13. Let

$T_1 =$

a_1		0	
a_1	b_2	b_1	b_3
b_4	b_2	0	b_3
a_1	b_2	b_6	b_7

$T_2 =$

a_4		a_5	
a_4	b_8	b_9	a_6
b_{10}	b_8	a_5	a_6

Both T_1 and T_2 are simple tableaux. The third row of T_1 is covered by the first row of T_1 or the first row of T_2. No row of T_1 covers the second row of T_2. □

LEMMA 5. *Let T_1 and T_2 be two simple tableaux without any repeated nondistinguished variables. Then $T_1 \equiv T_2$ if and only if T_1 and T_2 have identical summaries (up to renaming of distinguished variables), every row of T_1 is covered by some row of T_2, and every row of T_2 is covered by some row of T_1.*

Proof (If). We can map each row of T_1 to a row of T_2 that covers it. As there are no repeated nondistinguished variables, this mapping is a containment mapping. Thus $T_1 \supseteq T_2$. In the same way, $T_2 \supseteq T_1$, so $T_1 \equiv T_2$.

(Only if). A containment mapping θ from T_1 to T_2 surely maps distinguished variables to distinguished variables and constants to identical constants. Thus for every row r of T_1, r is covered by $\theta(r)$. The argument for the rows of T_2 is the same. □

6.3. Row closures. Suppose that T is a simple tableau. Let S be the set of all the rows of T that contain a repeated nondistinguished variable in one particular column. Let w be any row of T. The *closure* of S with respect to w, denoted $CL_w(S)$, is the minimal set of rows that contains S and satisfies the following condition:

> if x_1 is in $CL_w(S)$ and x_2 is any row of T such that x_1 and x_2 have the same repeated nondistinguished variable in some column, and w has a different symbol in this column, then x_2 is in $CL_w(S)$.

LEMMA 6. *Let T be a simple tableau and S the set of rows of T that contain a repeated nondistinguished variable in column A. Let w be a row of T, and let θ be defined by*

$$\theta(x) = \begin{cases} w & \text{for all } x \text{ in } CL_w(S), \\ x & \text{otherwise.} \end{cases}$$

Then θ is a containment mapping of T to T if and only if w covers $CL_w(S)$.

Proof (*Only if*). This portion is immediate from the definition of a containment mapping and of the covering relation.

(*If*). Suppose not. By the definition of the covering relation, we know that θ maps distinguished variables to distinguished variables, and constants to identical constants. Thus there exist rows y and z of T such that y and z have the same symbol d in some column B, and differing symbols in rows $\theta(y)$ and $\theta(z)$ of column B. Let us consider three cases.

Case 1. Neither y nor z is in $CL_w(S)$. Clearly $\theta(y)$ and $\theta(z)$ have the same symbol, d, in column B, so no violation occurs.

Case 2. y is in $CL_w(S)$ and z is not (or vice-versa). It is not possible that d is a nondistinguished variable, for it is repeated, and then by the definition of closure, z would be in $CL_w(S)$. (Note that $w = \theta(y)$ must differ in column B from $z = \theta(z)$, so w does not have d in column B.) If d is a distinguished variable or constant, then as w covers $CL_w(S)$ and y is in $CL_w(S)$, it follows that w has d in column B. But then $\theta(y) = w$ and $\theta(z) = z$ each have the same symbol d in column B, contrary to assumption.

Case 3. y and z are in $CL_w(S)$. Then $\theta(y) = \theta(z) = w$, so no violation of the containment mapping condition can be found. □

Let us define a *w-chain* to be a sequence of rows $z_1, z_2, \cdots, z_k$, $k \geqq 1$, such that for $1 \leqq i < k$, there is some column in which z_i and z_{i+1} have the same nondistinguished variable, and in which w does not have this variable. Then, by definition of closure, z_1 is in $CL_w(S)$ if and only if there exists a w-chain $z_1, z_2, \cdots, z_k$, such that z_k is in S.

LEMMA 7. *Suppose A and B are two columns of a simple tableau T with repeated nondistinguished variables in sets of rows* S_1 *and* S_2, *respectively. Suppose x covers* $CL_x(S_1)$ *and* θ_1 *is the containment mapping that sends* $CL_x(S_1)$ *to x and other rows to themselves. Let* T' *be* T *with the rows of* $CL_x(S_1) - \{x\}$ *eliminated. Let* $S_3 = S_2 - (CL_x(S_1) - \{x\})$. *It follows that if* S_3 *contains two or more rows, and in* T', *w covers* $CL_w(S_3)$, *then in T, w covers* $CL_w(S_2)$.

Proof. Case 1. Suppose x is not in $CL_w(S_3)$ in T'. We prove by induction on the length of a w-chain in T from y to some z in S_2, that in T', y is in $CL_w(S_3)$.

Basis. Length = 1. Here y is in S_2, since it has the same nondistinguished variable as z in column B. Suppose y is not in S_3. Then y is in $CL_x(S_1)$. Since S_3 has at least two elements, we may assume that z is in $S_3 - \{x\}$ and, therefore, z is not in $CL_x(S_1)$. Then x must have the same nondistinguished variable as y and z in column B, else z would be in $CL_x(S_1)$. Therefore x is in S_2, and as x is certainly not in $CL_x(S_1) - \{x\}$, it follows that x is in S_3, a contradiction.

Induction. Let there be a chain of length $k > 1$, say $y = z_1, z_2, \cdots, z_k = z$ from y to z. By the inductive hypothesis, z_2 is in $CL_w(S_3)$ in T'. Now there is a column such that y and z_2 have the same nondistinguished variable, and w has a different symbol there. If x has the repeated nondistinguished variable in that column, then x is in $CL_w(S_3)$. As we assume x not to be in $CL_w(S_3)$, if y is in $CL_x(S_1)$, then z_2 is in $CL_x(S_1)$, and therefore not in $CL_w(S_3)$. It follows that y is present in T' and therefore in $CL_w(S_3)$ in T'. Thus w covers $CL_w(S_2)$ in T, and the lemma follows.

Case 2. x is in $CL_w(S_3)$ in T'. Let θ_2 be the containment mapping on T' that sends members of $CL_w(S_3)$ to w and other rows of T' to themselves. Then $\theta_2\theta_1$ is a containment mapping on T. We claim that $\theta_2\theta_1$ maps all of $CL_w(S_2)$ to w. Let y be in $CL_w(S_2)$. If y is in $CL_x(S_1)$, then θ_1 maps y to x, and θ_2 maps x to w.

If y is not in $CL_x(S_1)$ but is in $CL_w(S_2)$, then there is in T a w-chain $y = z_1, z_2, \cdots, z_n$, where z_n is in S_2. An induction similar to the one above shows that y is in $CL_w(S_3)$ in T'. We prove the inductive step. If z_2 is in $CL_w(S_3)$, then so is y. Therefore

assume z_2 is in $CL_x(S_1)-\{x\}$. Consider the column in which y and z_2 have some nondistinguished variable, and w differs. If x does not have the repeated nondistinguished variable in this column, then y is in $CL_x(S_1)$. But in the opposite case, as x is in $CL_w(S_3)$, it follows that y is also, proving the induction.

As $\theta_2\theta_1$ is a containment mapping that sends all of $CL_w(S_2)$ to w, it must be that w covers $CL_w(S_2)$ in T. The present lemma then follows from Lemma 6. □

Consider again a simple tableau T. Let S be the set of all the rows with a repeated nondistinguished variable in a particular column. If we can find a row w that covers every row in $CL_w(S)$, then we can reduce T to an equivalent tableau T' by deleting all the rows of $CL_w(S)$ (except w, if w is in S) from T. This reduction rule can be applied repeatedly, to any column of T' that has a repeated variable, until we get a tableau that cannot be reduced further.

Example 14. Let

$T =$

a_1	a_2			
a_1	a_2	b_1	b_2	b_3
a_1	b_4	b_7	b_5	b_3
b_6	a_2	b_7	b_2	b_8
b_9	a_2	b_{10}	b_{11}	b_3

T is a simple tableau. Let $S=\{1, 3\}$[2] be the set of all the rows with variable b_2. $CL_1(S)=\{1, 2, 3\}$. That is, we begin with $CL_1(S)=S=\{1, 3\}$. Then, as row 2 has in column 3 the same repeated nondistinguished variable as row 3, but row 1 does not have this symbol, we add 2 to $CL_1(S)$. Row 4 has only distinguished variables and nonrepeated nondistinguished variables, except in column 5. But rows 1 and 4 have the same symbol there, so we cannot add 4 to $CL_1(\{1, 2, 3\})$.

The first row covers every row in this closure and, therefore, T can be reduced to

a_1	a_2			
a_1	a_2	b_1	b_2	b_3
b_9	a_2	b_{10}	b_{11}	b_3

.

Now, consider all the rows with the repeated variable b_3—these are all the remaining rows, and the first row covers them. Thus the above tableau is reduced to

a_1	a_2			
a_1	a_2	b_1	b_2	b_3

.

6.4. Promotion of repeated nondistinguished variables. We shall now prove that if for no w can $CL_w(S)$ be eliminated by Lemma 6, then the repeated nondistinguished variable that gave rise to S can be promoted to a constant.

[2] 1 means the first row, etc.

LEMMA 8. *Let T_1 and T_2 be simple tableaux, and let A be a column with a repeated nondistinguished variable b_1 appearing in set of rows S of T_1. Suppose also that there is no row w such that w covers $CL_w(S)$. Then*:

(a) *If $T_1 \equiv T_2$, then there is a repeated nondistinguished variable b_2 in the A column of T_2.*

(b) *If b_2 exists, and T_1' and T_2' are the tableaux that result from T_1 and T_2 by replacing b_1 and b_2 by the same constant, a constant that appears nowhere else, then T_1' and T_2' are simple, and $T_1 \equiv T_2$ if and only if $T_1' \equiv T_2'$.*

Proof (a). As $T_1 \equiv T_2$, let θ_1 and θ_2 be containment mappings from T_1 to T_2 and back, respectively. Let S' be the set of rows of T_2 such that $S' = \{\theta_1(w) | w \text{ is in } S\}$, and let $S'' = \{\theta_2 \cdot \theta_1(w) | w \text{ is in } S\}$. Then S'' has two or more members, and so does S'. The rows of S' have some one symbol d in column A. If d is a distinguished variable or constant, then as θ_2 is a containment mapping, the rows of S'' all have a distinguished variable or constant in column A. As there are at least two rows in S'', we violate our assumption that T_1 is simple. Therefore d is a repeated nondistinguished variable of T_2.

(b) We know containment mappings between T_1 and T_2 exist, if $T_1 \equiv T_2$. If these mappings did not map b_1 to b_2 and vice-versa then there would be a containment mapping from T_1 to itself that mapped S to one row, since no repeated symbols but b_1 and b_2 exist in their columns. We would thus violate our assumption that no w covers $CL_w(S)$. It follows that the containment mappings between T_1 and T_2 also serve for T_1' and T_2'. Conversely, containment mappings between T_1' and T_2' surely serve for T_1 and T_2. The fact that T_1' and T_2' are simple is obvious. □

6.5. The algorithm. We say a simple tableau is in *reduced form* if it has no repeated nondistinguished variables. Lemmas 6 and 8 can be used to put simple tableaux in reduced form, and Lemma 5 can be used to test the equivalence of two such tableaux. The algorithm is summarized in Fig. 5. The procedure REDUCE(T_1, T_2) puts T_1 in reduced form and also returns **false** if $T_1 \not\equiv T_2$ is detected. REDUCE returns **true** if it does not detect that $T_1 \not\equiv T_2$; note that T_1 may still not be equivalent to T_2 in this case.

THEOREM 10. *The algorithm of Fig. 5 correctly decides the equivalence of simple tableaux in $O(s^3t^2)$ time if the tableaux have a maximum of s rows and t columns.*

Proof. Lines (1)–(5) apply Lemma 6. The only important detail is that after looking at each column A and row w once, we need not reconsider A and w if they fail the test of line (4) once. In proof, note that by Lemma 7 applied once for each application of Lemma 6, no new opportunities for reduction are created as reductions are made.

Lines (6)–(10) implement Lemma 8, so the resulting T_1 is in reduced form. The test of line (14) then decides the issue by Lemma 5, if line (7) has not already detected that $T_1 \not\equiv T_2$.

For the running time of Fig. 5, we note that the loop of lines (1)–(5) is executed st times. Computation of $CL_w(S)$ at line (4) takes time $O(s^2t)$, since $O(st)$ is sufficient to check if any rows can be added to the closure, and at most s rows can be added. Thus the loop of (1)–(5) takes $O(s^3t^2)$ time. Clearly $O(st)$ time suffices for the loop of (6)–(10), so REDUCE takes $O(s^3t^2)$ time.

In the main procedure, lines (12) and (13) take $O(s^3t^2)$ time by the foregoing argument. Line (14) takes $O(s^2t)$ time, so the entire algorithm takes $O(s^3t^2)$ time. □

COROLLARY. *If n is the size of the input (i.e., n is the space needed to write down T_1 and T_2), then the algorithm of Fig. 5 takes $O(n^3)$ time.*

Proof. Note that st could be replaced by n in the above analysis, and $s \leqq n$ is obvious. □

```
     procedure REDUCE(T1, T2);
     begin
(1)      for each column A of T1 and row w of T1 do
(2)          if A has a repeated nondistinguished variable b then
                 begin
(3)                  let S be the set of rows in which b appears;
(4)                  if w covers CLw(S) then
(5)                      remove the rows in CLw(S)−{w} from T1
                 end
(6)      for each column A of T1 in which a repeated
             nondistinguished variable b1 remains do
             begin
(7)              if the column for A in T2 has no repeated nondistinguished
                 variable then
(8)                  return false; /*T1 ≰ T2*/
(9)              let b2 be the repeated nondistinguished variable in column A of T2;
(10)             make b1 and b2 be the same new constant;
             end;
(11)     return true
     end REDUCE;
     begin /* main procedure */
(12)     if ¬REDUCE(T1, T2) then return false;
         /* as a side effect, T1 is reduced */
(13)     if ¬REDUCE(T2, T1) then return false:
             /* as a side effect, T2 is reduced */
             /* note that lines (6)–(10) of REDUCE are not needed here */
(14)     if every row of T1 is covered by a row of T2, and vice versa then
(15)         return true
         else
(16)         return false
     end
```

FIG. 5. *Polynomial algorithm to test equivalence of simple tableaux.*

Note that the coverage of each row of T_1' by a row T_2' is a sufficient, but not a necessary, condition for $T_2' \subseteq T_1'$ (even when both T_1' and T_2' are in reduced form). Also observe that the results of Section 5 imply that containment is NP-complete for simple tableaux.

7. Extension to strong equivalence. The equivalence and containment results of the previous sections also apply to strong equivalence. We shall state these results here without proof. In each case the proof is analogous to that of the corresponding result about weak equivalence. We can use a modified form of tableau to represent values of expressions as mappings from their operands, rather than from an instance of the universe. The modifications that must be made are:

(1) rows are *tagged* with the relation from which they come,

(2) rows have blanks in columns corresponding to attributes that are not part of the relation with which the row is tagged.

Suppose T is such a tableau, with set of symbols S, summary w_0 and rows $w_1, w_2, \cdots, w_n$. Suppose $R_1, R_2, \cdots, R_k$ are the available relation schemes, and

$r_1, r_2, \cdots, r_k$ are corresponding relations. Let w_i be tagged by R_{j_i}, for $1 \leqq i \leqq n$. Then

$$T(r_1, r_2, \cdots, r_k) = \{\rho(w_0) | \text{for some } \rho : S \to D \text{ we have } \rho(w_i) \text{ in } r_{j_i} \text{ for } 1 \leqq i \leqq n\}.$$

7.1. The strong equivalence test. Tagged tableaux can be constructed from expressions exactly as in Theorem 1. The only modification is that the tableau for a relation scheme R has blank, rather than a nondistinguished symbol, in columns that do not correspond to attributes of R. We shall state the following analog of Corollary 1 to Theorem 2.

THEOREM 11. *Two tableaux are strongly equivalent if and only if containment maps that preserve tags exist in both directions.*

Example 15. Consider the expression $E = \pi_{AB}(AB \bowtie BC)$ from Example 3. The tagged tableau for AB is

A	B	C	
a_1	a_2		
a_1	a_2		(AB)

and for BC it is

A	B	C	
	a_2	a_3	
	a_2	a_3	(BC)

The tagged tableau for $AB \bowtie BC$ is

A	B	C	
a_1	a_2	a_3	
a_1	a_2		(AB)
	a_2	a_3	(BC)

and for E it is

A	B	C	
a_1	a_2		
a_1	a_2		(AB)
	a_2	b_1	(BC)

Note that a tag-preserving containment mapping from the tableau for AB to the above tableau exists, implying that $AB \supseteq \pi_{AB}(AB \bowtie BC)$ in the strong sense. However, no tag-preserving containment mapping exists in the other direction, since the tableau for AB has no row tagged (BC). Thus E is not strongly equivalent to AB, although we saw in Example 5 that these expressions are weakly equivalent. □

7.2. Functional dependencies. We may apply functional dependencies to tagged tableaux exactly as in Theorem 6. The two rows involved need not have the same tag,

provided we understand that functional dependencies apply to two or more relations jointly. For example, suppose that ABC and ABD are relation schemes, and $A \to B$ is a functional dependency. Then it is not permissible to have $a_1b_1c_1$ in ABC and $a_1b_2d_1$ in ABD. If we do not make this prohibition, then functional dependencies may only be applied to rows with the same tag.

7.3. Polynomial-time reductions between weak and strong equivalence. We can prove general results which show that the questions of weak and strong equivalence are almost the same problem.

LEMMA 9. *Let E_1 and E_2 be expressions. Then in time polynomial in the size of E_1 and E_2 we can construct expressions E_1' and E_2' such that E_1' and E_2' are strongly equivalent if and only if E_1 and E_2 are weakly equivalent.*

Proof. Let $R_1, R_2, \cdots, R_k$ be all the arguments of E_1 and E_2, and let $R = \bigcup_{i=1}^{k} R_i$. Construct E_1' and E_2' by replacing each operand R_i by $\pi_{R_i}(R)$. Then T behaves as a univeral relation, and a proof that E_1' is strongly equivalent to E_2' if and only if E_1 and E_2 are weakly equivalent is immediate from definitions. □

LEMMA 10. *Let E_1 and E_2 be expressions. Then in time polynomial in the size of E_1 and E_2 we may construct expressions E_1' and E_2' that are weakly equivalent if and only if E_1 and E_2 are strongly equivalent.*

Proof. Let G be a new attribute and let $R_1, R_2, \cdots, R_k$ be the operands of E_1 and E_2. Let $R_i' = R_i \cup \{G\}$ for all i. Construct E_1' and E_2' from E_1 and E_2 by replacing operand R_i by $\pi_{R_i}(\sigma_{G=i}(R_i'))$. Then the projection of the universal instance onto R_i', followed by selection of $G = i$ and projection to remove the G column yields a relation that is independent of any other relation derived from that instance by selection of another value of G. □

7.4. Complexity results for strong equivalence.

THEOREM 12. *Strong equivalence is NP-complete in each of the following cases.*

(i) *Tableaux are not required to come from expressions but may not have constants, nor may there be functional dependencies.*

(ii) *Tableaux are permitted to have constants, but must come from expressions and there may be no functional dependencies.*

(iii) *Functional dependencies are permitted, but tableaux must come from expressions and may not have constants.*

Proof. The construction of Lemma 9 preserves the absence of constants and the property that a tableau comes from an expression. The absence of functional dependencies is surely preserved. Note that the construction of Lemma 9 may be applied to tableaux as well as expressions, by simply filling out blanks in rows by new nondistinguished variables, so part (i) has meaning. The theorem then follows immediately from Theorems 7, 8, and 9. □

THEOREM 13. *Strong equivalence is decidable in polynomial time for expressions that have simple tableaux.*

Proof. The construction of Lemma 10 preserves simplicity of tableaux, as the column for G has only constants. □

Let T be any tableau tagged with relation schemes. For each repeated symbol s, let TAG(s) be the set of tags of rows that contain s. A *global repeated nondistinguished variable* is a nondistinguished variable b such that TAG(b) contains two or more tags. A tableau is *quasi-simple* if the following hold.

(a) If b is a global repeated nondistinguished variable in column A, then for every repeated symbol s, $s \neq b$, in column A, TAG(b) $\not\subseteq$ TAG(s).

(b) For each tag the set of all the rows with this tag is a simple tableau.

Note that a simple tableau is also quasi-simple. Condition (a) implies that a global repeated nondistinguished variable cannot be eliminated by row covering, and therefore it can be promoted to a constant immediately. Using condition (b), we can now minimize each set of rows with the same tag separately, using the algorithm for simple tableaux.

This approach can also be used whenever a tableau has a pattern of constants and/or distinguished variables that decompose each tableau to several disjoint sets of rows, such that no rows in one set can be mapped to a row in any other set.

8. Conclusions and open problems. Using tableaux, we have developed a "crank" that can be turned to tell whether two expressions over the set of relational operators select project, and (natural) join, are equivalent. The "crank" is capable of accounting for the effect of functional dependencies and works for either weak or strong equivalence. Although the "crank" requires exponential time in the general case, we have isolated an important special case for which a polynomial time equivalence algorithm was developed.

We have not considered the natural next step, which is to develop tools for the efficient optimization of expressions, given an arbitrary cost criterion. Our NP-completeness results suggest that any method involving canonicalization of expressions is likely to require considerable computational effort for general expressions, so the optimization problem appears to be very hard. However, the following problems appear appropriate for examination.

(1) How far can we extend the class of expressions for which equivalence is efficiently decidable?

(2) Can the equivalence test be made to work in even exponential time when there are multivalued dependencies [7], [14], [15], [26] that must be satisfied? A doubly exponential algorithm follows from the techniques of [1] for multivalued dependencies.

(3) Find a complete axiom system to transform an expression into any equivalent one. Note that the number of steps needed to go between equivalent expressions might be polynomial in their size without violating the NP-completeness results or proving P = NP, as finding the right sequence of steps might be hard.

REFERENCES

[1] A. V. AHO, C. BEERI AND J. D. ULLMAN, *The theory of joins in relational databases*, Proc. 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 107–113.

[2] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[3] A. V. AHO, R. SETHI AND J. D. ULLMAN, *Code optimization and finite Church–Rosser systems*, Design and Optimization of Compilers, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 89–105.

[4] W. W. ARMSTRONG, *Dependency structures of data base relationships*, Proc. IFIP 74, North Holland, 1974, pp. 580–583.

[5] P. A. BERNSTEIN, *Synthesizing third normal form relations from functional dependencies*, ACM Trans. on Database Sys., 1 (1976), pp. 277–298.

[6] P. A. BERNSTEIN AND C. BEERI, *An algorithmic approach to normalization of relational database schemes*, TR CSRG-73, Computer Science Research Group, University of Toronto, Sept. 1976.

[7] C. BEERI, R. FAGIN AND J. H. HOWARD, *A complete axiomatization for functional and multivalued dependencies*, Proc. ACM SIGMOD International Conference on the Management of Data, August, 1977, pp. 47–61.

[8] A. K. CHANDRA AND P. M. MERLIN, *Optimal implementation of conjunctive queries in relational data bases*, Proc. Ninth Annual ACM Symposium on Theory of Computing, May, 1976, pp. 77–90.

[9] E. F. CODD, *A relational model for large shared data banks*, Comm. ACM 13, (1970), pp. 377–387.
[10] ———, *Further normalization of the data base relational model*, Data Base Systems, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 33–64.
[11] ———, *Relational completeness of data base sublanguages*, Data Base Systems, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 65–98.
[12] S. A. COOK, *The complexity of theorem proving procedures*, Proc. 3rd Annual ACM Symposium on Theory of Computing, May, 1971, pp. 151–158.
[13] C. J. DATE, *An Introduction to Database Systems*, second ed., Addison-Wesley, Reading, MA, 1977.
[14] C. DELOBEL, *Contributions théoretiques à la conception d'un système d'informations*,' Ph.D. thesis, Univ. of Grenoble, Oct., 1973.
[15] R. FAGIN, *Multivalued dependencies and a new normal form for relational data-bases*, ACM Trans. Database Sys., 2, (1977), pp. 262–278.
[16] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
[17] P. A. V. HALL, *Optimization of a single relational expression in a relational data-base system*, IBM J. Res. Develop., 20 (1976), pp. 244–257.
[18] R. M. KARP, *Reducibility among combinatorial problems*, Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–104.
[19] J. MINKER, *Performing inferences over relational databases*, Proc. ACM SIGMOD International Conference on Management of Data (May 1975, San Jose, California), pp. 79–91.
[20] F. P. PALERMO, *A database search problem*, Information Systems COINS IV, J. T. Tou, ed., Plenum Press, New York, 1974.
[21] R. M. PECHERER, *Efficient evaluation of expressions in a relational algebra*, Proc. ACM Pacific Conf., April, 1975, pp. 44–49.
[22] J. RISSANEN, *Independent components of relations*, ACM Trans. Database Sys., 2(1977), pp. 317–325.
[23] J. M. SMITH AND P. Y.-T. CHANG, *Optimizing the performance of a relational algebra database interface*, Comm. ACM, 18(1975), pp. 568–579.
[24] M. STONEBRAKER AND L. A. ROWE, *Observations on data manipulation languages and their embedding in general purpose programming languages*, 2 TR UCB/ERL M77-53, University of California, Berkeley, July 1977.
[25] E. WONG AND K. YOUSSEFI, *Decomposition—a strategy for query processing*, ACM Trans. Database Sys. 1, (1976), pp. 223–241.
[26] C. ZANIOLO, *Analysis and design of relational schemata for database systems*, Tech. Rept. UCLA-ENG-7769, Department of Computer Science, UCLA, July, 1976.
[27] M. M. ZLOOF, *Query-by-Example: the invocation and definition of tables and forms*, Proc. ACM International Conf. on Very Large Data Bases, Sept., 1975, pp. 1–24.