SOLVING QUERIES BY TREE PROJECTIONS

by

Y. Sagiv* and O. Shmueli**

Technical Report #399
February 1986

* Dept. of Computer Science, Stanford University, CA. USA
** Dept. of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel.
Solving Queries by Tree Projections

Yehoshua Sagiv

Department of Computer Science
Stanford University

Oded Shmueli

Computer Science Department,
Technion - Israel Institute of Technology, Haifa, Israel

Abstract

Suppose a database schema $D$ is extended to $\bar{D}$ by adding new relation schemas, and states for $D$ are extended to states for $\bar{D}$ by applying joins and projections to existing relations. It is shown that certain desirable properties that $\bar{D}$ has with respect to $D$ are equivalent to the existence of a tree projection of $\bar{D}$ with respect to $D$. These properties amount to the ability to efficiently compute the join of all the relations in a state for $D$ from an extension of this state over $\bar{D}$. The equivalence is proved for unrestricted (i.e., both finite and infinite) databases. If $\bar{D}$ is obtained from $D$ by adding a set of new relation schemas that form a tree schema, then the equivalence also holds for finite databases. In this case there is also a polynomial time algorithm for testing the existence of a tree projection of $\bar{D}$ with respect to $D$.

1. Introduction

Since its introduction by Bernstein, Chi and Goodman [BC, BG], acyclicity has emerged as a basic concept in relational database theory (cf. [ull]). Acyclic (i.e., tree) database schemas have many important properties that cyclic database schemas lack, and they were investigated in dependency theory and database design [BFMMUY, BFMY, FMU, Hul, Kat, LMG, SMF], as well as in query processing [Fag, GS1-GS4, GST, KY, KYY, OC, Yan, YQ].

Consider a query $Q$ which is the join of some relations projected on a prescribed set of attributes. If the schemas of these relations form a tree schema, then we can evaluate $Q$ efficiently, i.e., in time polynomial in the size of $Q$, the operand relations and the result [Yan]. Tree schemas, however, are not only a case where $Q$ can be efficiently evaluated; moreover, they must be "hidden" in any evaluation of $Q$. Formally, suppose $Q$ is solved by a program $P$, i.e., a finite sequence of steps such that in each step a new relation is created from existing ones by applying either project, join, or semijoin (the semijoin of a relation $R$ by a relation $S$ is the join of $R$ and $S$ projected onto the attributes of $R$). Program $P$ creates relations over new relation schemas. The relation computed in the last step is the desired result. It has been shown in [GS3] that $P$ must

---

1On leave of absence from Hebrew University, Jerusalem, Israel. Work partially supported by NSF grant IST-84-12791, a grant of AT&T Foundation, and a grant of IBM Corp.
create relations over new relation schemas, such that there is a tree database schema consisting of the relation schemas mentioned in Q, the relation schema of the result, and subsets of the new relation schemas created by P. This database schema is called a tree projection (precise definitions are given in Section 2).

In this paper we extend the result of [GS3] by allowing program P to include (as a last step) a "semijoin loop," i.e., a step where relations are reduced by repeatedly replacing a relation with its semijoin by another relation until no relation can be reduced anymore. We show that a program (with a "semijoin loop") for solving query Q must produce a tree projection. This result has been proved earlier for a restricted class of queries whose adjacency graphs\(^2\) are either simple cycles or cliques [Kla]. Our results are also an extension of previous work done for acyclic database schemas. Some desirable properties of database schemas have been shown to be equivalent to acyclicity [BFMY,GS1,YO], in particular, the property that pairwise consistency implies global consistency, and the property that there is a monotone join expression of the database schema. We show that suitable generalizations of these properties are equivalent to the existence of a tree projection. Some of these equivalences are proved assuming that databases are unrestricted; i.e., both finite and infinite databases are allowed. In Section 4 we consider an important special case in which all the equivalences hold for finite databases. For this special case we also give a polynomial time algorithm for testing the existence of a tree projection.

2. Terminology

2.1. Relational Databases

A universe \(U\) is a finite set of attributes. A relation schema \(R_i\) is a subset of \(U\), and a database schema \(D\) (or simply schema) is a multiset of relation schemas. Clearly, a database schema may be viewed as the set of edges of a hypergraph over \(U\) [Ber]. We assume that each \(A \in U\) has the integers as its domain of possible values. Let \(U(D)\) denote the set of all the attributes in schema \(D\).

A relation \(R_i\) for a relation schema \(R_i\) is a set of records or tuples, where each tuple is a mapping of the attributes of \(R_i\) to constants (i.e., integers). One can think about the relation \(R_i\) as a table of data with columns for the attributes of \(R_i\) and tuples as rows. A database state (or simply a state) for schema \(D\) is an assignment of relations to the relation schemas of \(D\). We use \(D = (R_1, \ldots, R_n)\) to denote a database schema and \(D = (R_1, \ldots, R_n)\) for a corresponding state. Tuple \(r\) over schema \(R\) matches tuple \(s\) over schema \(S\) if for all \(A \in R \cap S\), the values of tuples \(r\) and \(s\) for attribute \(A\) are identical.

\(^2\)An adjacency graph has attributes as nodes and edges whenever two attributes are in the same relation schema.
The projection of a relation \( R \) over a set of attributes \( X \subseteq R \), denoted \( R[X] \) or \( \pi_X(R) \), is the relation over the schema \( X \) consisting of all tuples that match some tuple in \( R \). The (natural) join of relations \( R \) and \( S \), denoted \( R \bowtie S \), is defined as the relation for the schema \( R \cup S \) containing all tuples that match a tuple in \( R \) and a tuple in \( S \). The (natural) semijoin of relations \( R \) and \( S \), denoted \( R \bowtie S \) is defined as the relation containing all tuples of \( R \) matching some tuple in \( S \). Equivalently, \( R \bowtie S = \pi_R(R \bowtie S) \). Given a database \( D \) over schema \( D \), we denote \( D \) as \( J(D) \).

2.2. Tableaux

Tableaux are a formalism for formulating queries over a database schema \( (R_1, \ldots, R_n) \). A tableau \([ASU]\) is a table, similar to a relation, with columns that correspond to the attributes of the universe and rows that are filled with variables. We denote a tableau either by a single letter or by explicitly specifying its rows, i.e., \((w_1, \ldots, w_n) / w\) where \( w_i \) are the rows of the tableau and \( w \) is a special row called the summary. The summary and rows consist of blanks and variables. Each variable appearing in the summary must also appear in some \( w_i \), and no variable appears in more than one column. The variables appearing in the summary are called distinguished variables, and other variables are called nondistinguished variables. Each row has a tag, which is one of the relation schemas \( R_i \), and it has variables exactly in the columns of its tag (i.e., all the other columns are blank).

We say that \( h \) is a mapping of a tableau \( T \) if \( h \) maps each variable of \( T \) to a constant. The result of applying \( h \) to \( w \), where \( w \) is either a row or the summary of \( T \), is a tuple over the attributes in which \( w \) is nonblank, and it is defined naturally as follows. The tuple \( h(w) \) maps attribute \( A \) to \( h(w(A)) \), where \( w(A) \) is the variable appearing in column \( A \) of \( w \). The mapping \( h \) is a valuation of \( T \) into a database \((R_1, \ldots, R_n)\) if \( h \) maps every row of \( T \) with tag \( R_i \) to a tuple of \( R_i \).

A tableau defines a mapping from databases to relations. Let \( D = (R_1, \ldots, R_n) \) be a database. The tableau \( T \) maps \( D \) to the relation \( T(D) \) defined as follows:
\[
\{ h(s) \mid s \text{ is the summary of } T \text{ and } h \text{ is a valuation of } T \text{ into } D \}
\]
The schema of \( T(D) \) is the set of attributes in which the summary of \( T \) has nonblank symbols.

A tableau (or more precisely, the rows of a tableau) may be converted to a database state by mapping each variable to a distinct constant. Since the one-to-one mapping that converts a tableau to a state is only a formalism, we shall refer to the tableau itself as a state instead of using a mapping.

2.3. Containment and Equivalence of Tableaux

Let \( S \) be a set of database states for the schema \( D = (R_1, \ldots, R_n) \). A tableau \( T_1 \) contains a tableau \( T_2 \) over \( S \), written \( T_2 \subseteq_S T_1 \), if for all states \( D \) in \( S \), \( T_2(D) \subseteq T_1(D) \). These tableaux are
equivalent over $S$, written $T_1 =_S T_2$, if for all states $D$ in $S$, $T_2(D) = T_1(D)$. If $S$ is omitted, then $S$ is assumed to be the set of all states.

A homomorphism $h$ of $T_1$ into $T_2$ is a mapping of the variables of $T_1$ into the variables of $T_2$ that preserves summary and rows, that is

1. if $w$ is a row of $T_1$, then $h(w)$ is a row of $T_2$ having the same tag as $w$, and
2. the distinguished variables of $T_1$ are mapped to the distinguished variables of $T_2$, i.e., if $w$ is the summary of $T_1$, then $h(w)$ is the summary of $T_2$.

If $T_2$ is viewed as a state, then a homomorphism is a valuation of $T_1$ into $T_2$ with the additional restriction that distinguished variables are mapped to distinguished variables. The existence of a homomorphism from $T_1$ into $T_2$ is a sufficient condition for $T_2 \subseteq T_1$, and it is also a necessary condition for $T_2 \subseteq T_1$ if the rows of $T_2$ form a state of $S' \{\text{ASU}\}$. The mapping of the rows of $T_1$ into the rows of $T_2$ induced by a homomorphism $h$ is called a containment mapping.

2.4. Expressions

A relational expression consists of relation schemas as operands, join (denoted $\Join$) as a binary operator, project (denoted $\pi_X$) as a unary operator, and parentheses. An expression $E$ with operands drawn from the database schema $D = (R_1, \ldots, R_n)$ is (similarly to a tableau) a mapping from states of $D$ to relations. A state $D = (R_1, \ldots, R_n)$ is mapped to the relation $E(D)$ defined as follows. Assign $R_i$ to the operand $R_i$, and evaluate the expression $E$ according to the definition of the operators; the result is $E(D)$. For each expression there is a corresponding tableau that represents the same mapping [ASU].

We now describe the tableaux for two expressions that will be used later. The tableau for the expression $R_j$ has a single row with tag $R_j$ and all its variables are distinguished. The tableau for the expression $\pi_X(\bigwedge_{j=1}^n R_j)$ has one variable per column (with possibly multiple occurrences) and a row with tag $R_j$ (for $j = 1, \ldots, n$); the distinguished variables are those in the columns of $X$.

2.5. Inclusion Dependencies

An inclusion dependency (abbr. ID) is a statement of the form $R_i[X] \subseteq R_j[X]$, where $R_i$ and $R_j$ are relation schemas and $X$ is a set of attributes contained in both of them. The ID $R_i[X] \subseteq R_j[X]$ is satisfied by a state $D = (R_1, \ldots, R_n)$ if $R_i[X] \subseteq R_j[X]$, i.e., for each tuple of $R_i$ there is a tuple of $R_j$ with the same values for $X$. Note that $X$ may be the empty set and then the ID $R_i[X] \subseteq R_j[X]$ expresses the statement "if $R_i$ is not empty, then $R_j$ is not empty." Given a set of IDs $I$, the set of all states (of the database schema at hand) satisfying $I$ is denoted as $SA(T[I])$. 

Technion - Computer Science Department - Technical Report CS0399 - 1986
2.6. Finite vs. Unrestricted States

We usually assume (unless stated otherwise) that all states are finite, i.e., consist only of finite relations. Some of our results, however, hold only if we assume that states are unrestricted, i.e., infinite states are allowed as well as finite ones. When we state a result without explicitly referring to either the finite or unrestricted case, then it holds for both. Similarly, tableaux may also have an infinite number of rows. The definitions and results stated so far need not be changed when infinite tableaux and unrestricted states are considered.

2.7. Tree Schemas

Let \( G = (V, E) \) be an undirected graph\(^4\) whose nodes are in one-to-one correspondence with the relation schemas of a database schema\(^4\) \( D \). We say that \( G \) is \( A \)-connected if the subgraph of \( G \) induced by relation schemas (nodes) containing attribute \( A \) is connected. \( G \) is \( X \)-connected if for all \( A \in X \), the graph \( G \) is \( A \)-connected. \( G \) is a qual graph for \( D \) if it is \( \cup(D) \)-connected \([BG]\). We call this property of a qual graph attribute connectivity. A qual graph for schema \( D \) is minimal if there exists no other qual graph for \( D \) with a smaller number of edges:

\( D \) is a tree schema (acyclic hypergraph) if some qual graph for it is a tree; otherwise \( D \) is a cyclic schema (cyclic hypergraph). An attribute \( A \in \cup(D) \) is isolated if it appears in exactly one relation schema. The simple procedure of Figure 1, called Graham reduction and discovered independently by \([Gra]\) and \([YO]\), leads to a recognition of tree schemas. We denote by \( GR(D) \) the

---

**procedure** \( GR \) (\( D \) : database schema)

Apply the following two steps until neither is applicable:

**Step 1:** Delete any isolated attribute.

**Step 2:** Find two relation schemas \( R \) and \( S \) in \( D \) such that \( R \subseteq S \); delete \( R \) from \( D \).

**Figure 1.**

---

\(^4\)We use traditional graph theory notation.

\(^5\)When no confusion may arise we let \( V = D \) and talk about "node \( R \)" rather than "node \( v \in V \) corresponding to \( R \in D \)."

---

Technion - Computer Science Department - Technical Report CS0399 - 1986
output of procedure $GR$ on input $D$. It can be shown that $D$ is a tree schema if and only if upon termination of this procedure the resulting schema consists of a single empty relation schema, i.e. $GR(D) = \emptyset$. (A linear time algorithm for recognizing tree schemas appears in [TY].)

Example 2.1: The following is a tree schema: $D_1 = \{(1,2),(1,4),(1,5),(2,3),(1,2,3),(1,4,5)\}$. A qual graph for $D_1$ is:

$$\begin{align*}
&\begin{array}{c}
1,2 \\
\downarrow \\
1,2,3 \\
\downarrow \\
1,4,5 \\
\downarrow \\
1,4 \\
\end{array} \\
&\begin{array}{c}
2,3 \\
\downarrow \\
1,5 \\
\end{array}
\end{align*}$$

The schema $D_2 = \{(1,2),(2,3),(3,4),(4,1)\}$ is cyclic. Any minimal qual graph for $D_2$ is isomorphic to:

$$\begin{align*}
&\begin{array}{c}
1,2 \\
\downarrow \\
2,3 \\
\downarrow \\
1,4 \\
\end{array} \\
&\begin{array}{c}
3,4 \\
\end{array}
\end{align*}$$

2.8. Tree Projections

$D$ is a projection of $\overline{D}$, denoted $D \leq \overline{D}$, if for every relation schema $R \in \overline{D}$ there exists a relation schema $\overline{R} \in D$ such that $R \subseteq \overline{R}$. Let $D$ and $\overline{D}$ be database schemas such that $D \leq \overline{D}$. We say that $D_0$ is a tree projection (abbr. TP) of $\overline{D}$ w.r.t. (with respect to) $D$ if $D_0$ is a tree schema and $D \leq D_0 \leq \overline{D}$ [GS3]. If $\overline{D}$ is a tree schema, then obviously $\overline{D}$ is a TP (of itself) w.r.t. $D$. If $D_0$ is a tree projection of $\overline{D}$ w.r.t. $D$, then we may assume that $D \subseteq D_0$, since we can always add the relation schemas of $\overline{D}$ to $D_0$ and the result remains both a tree schema and a projection of $\overline{D}$, because $D \leq D_0$. A relation schema of the tree projection $D_0$ is called a base relation schema if it is also in $D$; otherwise, it is called a non-base relation schema.

Example 2.2:

Let $D = \{(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8),(8,1)\}$

$D_0 = D \cup \{(1,2,3,8),(3,4,7,8),(4,5,6,7)\}$

$\overline{D} = \{(1,2,5,6),(1,2,3,5,8),(1,3,4,7,8),(4,5,6,7),(5)\}$

Clearly both $D$ and $\overline{D}$ are cyclic, and $D_0$ is a tree schema. $D \leq D_0 \leq \overline{D}$, and therefore $D_0$ is a TP of $\overline{D}$ w.r.t. $D$.

3. Solving a Database-Schema

3.1. An Optimization Problem

Our goal is to compute $J(D)[X]$ for any given $X \subseteq \cup(D)$. If $D$ is a tree schema, then this can
be done in time polynomial in the size of the database schema, the state $D$ and the result [Yan].
On the other hand, for arbitrary database schemas, merely determining whether a tuple $t$ is in $\mathcal{J}(D)[X]$ is NP-complete [Yan]. Therefore, unless $P = NP$ there is no algorithm that computes $\mathcal{J}(D)$ in time polynomial in the size of the input $D$ and the result $\mathcal{J}(D)$. In particular, computing $\mathcal{J}(D)$ by a join expression (i.e., an expression consisting of the $R_i$, the join operator, and parentheses) is not a polynomial time process. This observation follows from the fact that intermediate results may be much larger than the final one (i.e., they may contain tuples that are not part of any tuple in $\mathcal{J}(D)$).

Suppose we compute $\mathcal{J}(D)[X]$ by evaluating a join expression $E$ to obtain $\mathcal{J}(D)$ and then projecting onto $X$. Let $R_1, \ldots, R_n, S_1, \ldots, S_k$ be the sequence of the original operands of $E$ (i.e., the $R_i$) and the intermediate results $S_1, \ldots, S_k$; the $S_i$ appear in the order in which they are computed during the evaluation of $E$. We denote as $R_1, \ldots, R_n, S_1, \ldots, S_k$ the relation schemas of $R_1, \ldots, R_n, S_1, \ldots, S_k$. If for some $m,(R_1, \ldots, R_n S_1, \ldots, S_m)$ is a tree schema, then we can apply the algorithm of [Yan] to compute $\mathcal{J}(D)[X]$ from $(R_1, \ldots, R_n S_1, \ldots, S_m)$ without having to actually compute $\mathcal{J}(D)$. Usually, applying the algorithm of [Yan] as soon as possible is more efficient than evaluating $\mathcal{J}(D)$ and projecting onto $X$, since $\mathcal{J}(D)$ may be much larger than $\mathcal{J}(D)[X]$. It is wrong, however, to assume that $(R_1, \ldots, R_n S_1, \ldots, S_m)$ must be a tree schema in order to apply the algorithm of [Yan]. We will prove that it is sufficient that there is a TP (i.e., tree projection) of $(R_1, \ldots, R_n S_1, \ldots, S_m)$ w.r.t. $(R_1, \ldots, R_n)$, i.e., that there is a tree schema $(R_1, \ldots, R_n, Q_1, \ldots, Q_m)$, where each $Q_i$ is a subset of some $S_j$. Note that it is quite possible that there is a TP of $(R_1, \ldots, R_n S_1, \ldots, S_m)$ w.r.t. $(R_1, \ldots, R_n)$, although there is no way to obtain a tree schema from $(R_1, \ldots, R_n S_1, \ldots, S_m)$ simply by deleting some of the $S_i$. In essence, we address the following problem. Given that some intermediate results have already been created, is it now possible to efficiently complete the computation of $\mathcal{J}(D)[X]$? We first define the necessary concepts needed to formalize this question.

### 3.2. Equivalence of Reduction Predicates

Schema $\mathcal{D} = (R_1, \ldots, R_n S_1, \ldots, S_m)$ is an extension of schema $D = (R_1, \ldots, R_n)$ if $\cup(D) = \cup(D)$. Let $D = (R_1, \ldots, R_n)$ be a state for $D$ and $\mathcal{D} = (R_1, \ldots, R_n S_1, \ldots, S_m)$ a state for $\mathcal{D}$. The state $\mathcal{D}$ is an extension of $D$ if $\mathcal{J}(D)[S_i] \subseteq S_i$ (for all $i = 1, \ldots, m$). Note that if each $S_i$ is the join of some of the relations $R_1, \ldots, R_n$ followed by a projection onto $S_i$, then $\mathcal{D}$ is an extension of $D$.

*Recall that $\mathcal{J}(D)$ is the join of all the $R_i$ in $D$.

The input consists of a database schema $\mathcal{D}$ and a state $\mathcal{D}$ for $\mathcal{D}$. 

7
State $D = (R_1, \ldots, R_n)$ is join reduced, denoted $\text{JR}(D)$, if for all $R_i$ in $D$, we have $R_i = \text{JR}(D)[R_i]$. State $D$ is semijoin reduced, denoted $\text{SJR}(D)$, if $R_i = R_i \times R_j$ (for all $i, j = 1, \ldots, n$). The reduction of state $D$, denoted $\text{red}(D)$, is a state $(R'_1, \ldots, R'_n)$ such that:

1. $R'_i \subseteq R_i$ (for all $1 \leq i \leq n$),
2. $\text{SJR}(R'_1, \ldots, R'_n)$, and
3. if $(R_{i1}^1, \ldots, R_{in}^n)$ satisfies (1) and (2), then $R'_i \subseteq R'_i$ (for all $1 \leq i \leq n$).

Note that the reduction of $D$ is unique, because if two distinct states satisfy (1) and (2), then so does the state whose relation for each $R_i$ is the union of the relations for $R_i$ in those two states. The reduction of a finite state $D$ can be computed by repeatedly applying semijoins.

**Proposition 1:** Let $D = (R_1, \ldots, R_n)$ be a state, $D' = (R_1, \ldots, R_m S_1, \ldots, S_m)$ an extension of $D$, and $D'' = (R'_1, \ldots, R'_n S'_1, \ldots, S'_m)$ the reduction of $D'$. Then $\text{JR}(D) = \text{JR}(R'_1, \ldots, R'_n)$.

**Proof:** Clearly, $\text{JR}(D) \supseteq \text{JR}(R'_1, \ldots, R'_n)$. We have to show that $\text{JR}(D) \subseteq \text{JR}(R'_1, \ldots, R'_n)$. Suppose not, i.e., there exists $t \in \text{JR}(D)$ such that $t \notin \text{JR}(R'_1, \ldots, R'_n)$. Obviously, $t[R_i] \in R_i$ (1 \leq i \leq n), and since $D$ is an extension of $D$, $t[S_i] \in S_i$ (1 \leq i \leq m). For $i = 1, \ldots, m$, let $s_i = t[S_i]$, and for $i = 1, \ldots, n$, let $r_i = t[R_i]$. Let

$$D^R = (R'_1 \cup \{r_1\}, \ldots, R'_n \cup \{r_n\}, S'_1 \cup \{s_1\}, \ldots, S'_m \cup \{s_m\})$$

By the above construction, $\text{SJR}(D^R)$ and, so, $D^R$ shows that $D'$ does not satisfy the third condition in the definition of the reduction of $D$. This contradiction proves $\text{JR}(D) \subseteq \text{JR}(R'_1, \ldots, R'_n)$.

**Proposition 2:** Consider schema $D$ and an extension $D'$. Suppose that $D = (R_1, \ldots, R_m S_1, \ldots, S_m)$ is an extension of $D = (R'_1, \ldots, R'_n)$. Then $D'' = (R'_1, \ldots, R'_n S'_1, \ldots, S'_m)$, the reduction of $D'$, is an extension of $(R'_1, \ldots, R'_n)$.

**Proof:** We have to show that for all $i = 1, \ldots, m$ the following holds

$$\left( \bigcup_{j=1}^{n} R'_j \right)[S_i] \subseteq S'_i \quad (A)$$

Suppose not, i.e., there exists $t \in \bigcup_{j=1}^{n} R'_j$ and $q (1 \leq q \leq m)$, such that $s_q = t[S_q]$ and $s_q \notin S'_q$. For $i = 1, \ldots, m$, let $s_i = t[S_i]$. Clearly, $s_i \in S'_i$. Let

$$D^R = (R'_1, \ldots, R'_n S'_1 \cup \{s_1\}, \ldots, S'_m \cup \{s_m\})$$

By the above construction, $\text{SJR}(D^R)$ and, so, $D^R$ shows that $D'$ does not satisfy the third condition.

---

\*We usually omit an extra pair of parentheses when there is no ambiguity, e.g., for a database state $(R'_1, \ldots, R'_n)$ we write $\text{JR}(R'_1, \ldots, R'_n)$ instead of $\text{JR}((R'_1, \ldots, R'_n))$.\
in the definition of the reduction of $D$. This contradiction proves (A). []

Let $D = (R_1, \ldots, R_n)$ be a database schema and let $\tilde{D} = (R_1, \ldots, R_n, S_1, \ldots, S_m)$ be an extension of $D$. We say that $\tilde{D}$ *solves* $D$ if for every state $D$ and an extension $\tilde{D}$ of $D$, the reduction $(R'_1, \ldots, R'_n, S'_1, \ldots, S'_m)$ of $D$ satisfies the predicate $JR(R'_1, \ldots, R'_n)$. Intuitively, when $\tilde{D}$ solves $D$, then $J[D][R'_i] (i=1,\ldots,n)$ can be obtained by computing an extension of state $D$ and then reducing it by semijoins to $(R'_1, \ldots, R'_n, S'_1, \ldots, S'_m)$, since $J[D][R'_i] = R'_i$ by Proposition 1 and the fact that $JR(R'_1, \ldots, R'_n)$.

Reducing an extension of $D$ by semijoins is polynomial in the size of the extension and, therefore, the problem of evaluating $J[D][R'_i]$ is considered solved when we find an extension from which $J[D][R'_i]$ can be obtained by semijoins. Solving $J[D][R'_i]$ in the obvious way of computing $J[D]$ and projecting it onto $R_i$ usually takes longer, and it always involves creating at some point an extension of $D$ from which $J[D][R'_i]$ can be obtained by semijoins. The optimization of computing $J[D][R'_i]$ requires, therefore, to determine whether $\tilde{D}$ solves $D$. Moreover, we will later show that if $\tilde{D}$ solves $D$, then we can do more than just efficiently compute $J[D][R'_i]$ from an extension $D$ of $\tilde{D}$, i.e., we can also efficiently compute $J[D]$ from $\tilde{D}$. Our goal is to characterize when $\tilde{D}$ solves $D$.

We now define two more concepts involving $D$ and $\tilde{D}$. Schema $\tilde{D}$ *strongly reduces* $D$ if $SJR(D)$ implies $JR(R_1, \ldots, R_n)$ for all states $\tilde{D} = (R_1, \ldots, R_n, S_1, \ldots, S_m)$. Schema $\tilde{D}$ *weakly reduces* $D$ if $SJR(D)$ implies $JR(R_1, \ldots, R_n)$ for all states $\tilde{D} = (R_1, \ldots, R_n, S_1, \ldots, S_m)$ which are extensions of $D = (R_1, \ldots, R_n)$. The definitions immediately imply that if $\tilde{D}$ strongly reduces $D$, then $\tilde{D}$ both weakly reduces and solves $D$. We also have the following.

**Theorem 1:** The following are equivalent for finite (and also for unrestricted) databases:

1. $D$ solves $D$.
2. $\tilde{D}$ strongly reduces $D$.
3. $\tilde{D}$ weakly reduces $D$.

**Proof:** (1) $\implies$ (2). We assume that $\tilde{D}$ solves $D$, and suppose that $\tilde{D}$ does not strongly reduce $D$, i.e., there exists a state $\tilde{D} = (R_1, \ldots, R_n, S_1, \ldots, S_m)$ for $\tilde{D}$ such that $SJR(D)$ but not $JR(R_1, \ldots, R_n)$. Let $D' = (R_1, \ldots, R_n, S'_1, \ldots, S'_m)$ be an extension of $D = (R_1, \ldots, R_n)$ such that $S'_i \supseteq S_i$ (we can obtain $D'$ by adding the tuples of each $S_i$ to the corresponding relation in any given extension). Consider the reduction $(R'_1, \ldots, R'_n, S'_1, \ldots, S'_m)$ of $D'$. Clearly, $R'_i \subseteq R_i$ and

---

*In fact, our results imply that when $\tilde{D}$ solves $D$, then the $(R'_1, \ldots, R'_n)$-part of the reduction can be obtained from the extension $\tilde{D}$ by applying a number of semijoins that depends only on the schemas $\tilde{D}$ and $D$, but not on the actual relations of $\tilde{D}$. *

*In the worst case this point is reached when $J[D]$ is computed, but in many cases it is reached when only some intermediate results have been evaluated.*
since \( D \) satisfies parts (1) and (2) in the definition of the reduction of \( D' \), we have \( R'_i \supseteq R_i \) and hence \( R'_i = R_i \). By (1), \( JR(R'_1, \ldots, R'_n) \), a contradiction.

(2) \( \implies \) (3). Obvious.

(3) \( \implies \) (1). Consider schemas \( D \) and \( D' \). Suppose that \( D = (R_1, \ldots, R_n, S_1, \ldots, S_m) \) is an extension of \( D = (R_1, \ldots, R_n) \). Let \( D' = (R'_1, \ldots, R'_n, S'_1, \ldots, S'_m) \) be the reduction of \( D \), by definition \( SJR(D) \). By Proposition 2, \( D' \) is an extension of \( (R'_1, \ldots, R'_n) \) and so, by (3), \( JR(R'_1, \ldots, R'_n) \), and thus \( D' \) solves \( D \).

3.3. Monotone Join expressions

Let \( D = (R_1, \ldots, R_n) \) be a database schema, and \( D = (R_1, \ldots, R_n, S_1, \ldots, S_m) \) an extension of \( D \). A join expression of \( D \) over \( D \) is a well-formed expression \( E \) consisting of relation schemas as operands, the join symbol \( \cdot \) as the only binary operator, and parentheses. All the schemas \( R_i \) must appear as operands of the join expression \( E \), and there may also appear additional operands \( Q_1, \ldots, Q_k \) provided that each \( Q_i \) is a subset of some \( S_j \). Given a state \( D = (R_1, \ldots, R_n, S_1, \ldots, S_m) \) such that \( SJR(D) \), the join expression \( E \) is evaluated as follows.

First, each \( R_i \) is assigned the relation \( R_i \); and each \( Q_i \) is assigned the relation \( S_j[Q_i] \), where \( S_j \) is some relation schema such that \( Q_i \subseteq S_j \) (since \( SJR(D) \), the relation assigned to \( Q_i \) is well-defined even if more than one \( S_j \) contains \( Q_i \)). Second, the join operators are applied to these relations as specified in the expression \( E \). Note that if \( D' \) is an extension of \( (R_1, \ldots, R_n) \), then the result is \( JR(R_1, \ldots, R_n) \).

The join expression \( E \) is monotone with respect to \( D \) if each join performed during its evaluation involves two relations \( P_i \) and \( P_j \) that are pairwise consistent, i.e., \( SJR(P_i, P_j) \) (note that each of these two relations is either an intermediate result or an original relation assigned to one of the operands).

We say that a join expression of \( D \) over \( D \) is monotone if it is monotone with respect to every state \( D = (R_1, \ldots, R_n, S_1, \ldots, S_m) \) such that \( SJR(D) \). A monotone join expression is an efficient process for completing the evaluation of \( J(D) \) once an extension \( D \) of \( D \) has been created, since the expression can be evaluated in time polynomial in the size of the input and the result \( J(D) \).

Consider the reduction \( (R'_1, \ldots, R'_n, S'_1, \ldots, S'_m) \) of an extension of a state \( D \). The condition that \( D \) solves \( D \) only implies that the reduction provides a solution for \( J(D)[R'_i] \), since \( J(D)[R'_j] = R'_i \). Having a monotone join expression of \( D \) over \( D \) implies that \( J(D) \) can be computed.

10The input consists of \( D, D' \), and the expression to be evaluated. In general the evaluation may take exponential time as a function of the size of the input and the result, but it takes polynomial time if the expression is monotone.
efficiently from the reduction, Thus, the latter condition is a priori stronger than the first one. We will show, however, that the two conditions are equivalent for unrestricted databases. Further, if there is a monotone join expression of \( D \) over \( D \), then we can use the algorithm of [Yan] in order to compute \( J(D)[X] \) from an extension \( D = (R_1, \ldots, R_n, S_1, \ldots, S_m) \) in time polynomial in the size of \( D \) and the result.

### 3.4. A Characterization of Strong Reducibility

In this section we show how to test whether \( D \) and \( D \) satisfy the predicates of Theorem 1. It is straightforward to reduce the question of whether \( D \) strongly reduces \( D \) to a tableau containment problem under some inclusion dependencies. This containment problem is decidable when databases are unrestricted, [JK], i.e., when infinite database states (as well as finite ones) are permitted. We can prove that when databases are unrestricted, this containment problem is equivalent to the existence of a TP of \( D \) w.r.t. \( D \). Since it is proved only for unrestricted databases, it follows that if there is no TP, then there is an infinite-counterexample state showing that \( D \) does not strongly reduce \( D \), and it is an open problem whether there is always a finite counterexample state as well. Later we will show that in some cases the existence of an infinite counterexample implies the existence of a finite one. Interestingly, in those cases for which we can show that there is a finite counterexample, we can also show that testing whether there is a TP can be done in polynomial time (in the size of the schemas alone), whereas in the general case we only know that this problem is in \( NP \).

Consider the expressions \( E_1 = (\prod_{i=1}^{n} R_i) R_i \) and \( E_2 = R_i \) and let \( T_1 \) and \( T_2 \) be their tableaux, respectively. By definition, the schema \( D = (R_1, \ldots, R_n, S_1, \ldots, S_m) \) strongly reduces \( D = (R_1, \ldots, R_n) \) if and only if for all \( 1 \leq i \leq n \), the expressions \( E_1 \) and \( E_2 \) are equivalent for all database states \( D \) for \( D \), such that \( SJR(D) \). The states \( D \) such that \( SJR(D) \), are exactly those states satisfying the set \( I \) of IDs (inclusion dependencies) defined as follows. Let \( P_1, \ldots, P_{n+m} \) be the same as \( R_1, \ldots, R_n, S_1, \ldots, S_m \). For each pair \( P_i \) and \( P_j \), the IDs \( P_i \cup P_j \subseteq P_i \) and \( P_i \cup P_j \subseteq P_j \) are in \( I \). Obviously, the expression \( E_1 \) is always contained in \( E_2 \) [ASU]. In order to test containment in the other direction, we have to chase [MMS] the tableau \( T_2 \) under the dependencies in \( I \). The chase process is based on a set of rules which generate new rows that are added to the tableau. The rule associated with a dependency \( P_i \subseteq P_i \) is as follows. (Note that \( X = P_i \cup P_i \)). If tableau \( T_2 \) has a row \( w \) tagged with \( P_i \), then we add a new row \( v \) tagged with \( P_i \) that has the same variables as \( w \) in the columns of \( X \) and new distinct variables in the rest of the columns of \( P_i \). The rules for the dependencies of \( I \) are applied repeatedly until no more rows can be added, and the final result is called the chase of \( T_2 \) w.r.t. \( I \), denoted \( chase(T_2) \).

It is convenient to represent \( chase(T_2) \) as an infinite ordered tree, where nodes correspond to rows. The root is the only row initially contained in \( T_2 \). For each node \( v \) children are created.
thus. Let \( P_i \) be the tag of \( v \). For each \( P_i \) we create a child of \( v \) having a tag \( P_i \) and variables as described above. We order the children of \( v \) from left to right according to the sequence \( P_1, \ldots, P_{n+m} \). Further, we create the whole tree in a breadth-first manner, i.e., each level (starting with the one below the root) is created from left to right, and each time a new variable is needed the next one is taken from an infinite sequence \( d_1, \ldots, d_i, \ldots \) of new variables. This process creates a well defined tableau whose rows can be viewed as an infinite state \( D \) such that \( SJR(D) \).

Now, \( T_1 \) contains \( T_2 \) over \( SAT(J) \) if and only if there is a homomorphism of \( T_1 \) into \( chase(T_2) \) [ASU].

**Theorem 2:** The following are equivalent when databases are unrestricted:

1. \( D \) solves \( D \).
2. \( D \) strongly reduces \( D \).
3. \( D \) weakly reduces \( D \).
4. There exists a TP of \( D \) w.r.t. \( D \).
5. There is a monotone join expression of \( D \) over \( D \).

**Proof:** Theorem 1 shows the equivalence of (1), (2) and (3) when databases are finite, and the same proof is also valid when databases are unrestricted. We will show that (4) \( \implies \) (5) and (5) \( \implies \) (3) when databases are either finite or unrestricted, and (2) \( \implies \) (4) when databases are unrestricted.

(4) \( \implies \) (5). Let \( D^t = (R_{11}, \ldots, R_{1n}, Q_{11}, \ldots, Q_{1k}) \) be a TP of \( D \) w.r.t. \( D \), i.e., \( D^t \) is a tree schema and each \( Q_i \) is a subset of some \( S_j \) in \( D \). Therefore, there is a monotone join expression \( E \) of \( D^t \) over \( D^t \), that is, the operands of \( E \) are all the \( R_i \) and \( Q_j \). It immediately follows that \( E \) is also a monotone join expression of \( D \) over \( D \), because if \( E \) is monotone with respect to every state \( D^t \) for \( D \) such that \( SJR(D^t) \), then, in particular, it is monotone with respect to any assignment \( R_{11}, \ldots, R_{1n}, Q_{11}, \ldots, Q_{1k} \) obtained from a state \( D \) such that \( SJR(D) \), since \( SJR(D) \) implies \( SJR(D^t) \).

(5) \( \implies \) (3). Let \( D = (R_11, \ldots, R_{rn}, S_1, \ldots, S_m) \) be an extension of state \( D = (R_{11}, \ldots, R_{1n}) \) (i.e., \( S_i \supseteq J(D)[S_i] \)), such that \( SJR(D) \). To prove the above claim we show that \( JR(R_{11}, \ldots, R_{1n}) \). Let \( E \) be a monotone join expression of \( D \) over \( D^t \) consisting of the operands \( R_{11}, \ldots, R_{1n}, Q_{11}, \ldots, Q_{1k} \) (each \( Q_i \) is a subset of some \( S_j \)). Construct an assignment of relations for the operands of \( E \) as follows. For \( i=1, \ldots, n \), the relation for \( R_i \) is the \( R_i \) obtained from \( D \). For \( i=1, \ldots, m \), the relation \( Q_i \) for \( Q_i \) is obtained by projection from some \( S_j \) such that \( S_j \supseteq Q_i \). Since \( S_j \supseteq \left( \bigcup_{i=1}^{n} R_i \right) \subseteq \left( \bigcup_{j=1}^{m} Q_j \right) \), it follows that \( Q_i \supseteq \left( \bigcup_{j=1}^{m} R_j \right) \subseteq \left( \bigcup_{j=1}^{m} Q_j \right) \). Therefore,

\[
J(D) = \bigcup_{i=1}^{n} R_i = (R_{11} \times \cdots \times R_{1n} \times Q_{11} \times \cdots \times Q_{1k})
\]

(A)

The right-hand side of (A) is the value of \( E \) for the above assignment, and since \( SJR(D) \) and \( E \) is monotone, it follows that \( R_i = J[R_i] \) for \( i=1, \ldots, n \) and, hence, \( JR(R_{11}, \ldots, R_{1n}) \). So, \( \bar{D} \) weakly...
(2) \implies (4). Suppose $D = (R_1, \ldots, R_n, S_1, \ldots, S_m)$ strongly reduces $D = (R_1, \ldots, R_n)$. Consider the expressions $E_1 = (\bigwedge_{i=1}^n R_i) \land R_1$ and $E_2 = R_1$, and their tableaux $T_1$ and $T_2$, respectively. Since $D$ strongly reduces $D$, tableau $T_1$ contains $\text{chase}(T_2)$ over $\text{SAT}(I)$. Therefore, there exists a homomorphism $h : T_1 \rightarrow \text{chase}(T_2)$. This mapping identifies a finite segment of the infinite tree representing $\text{chase}(T_2)$. This finite segment, which is also a tree, consists of the nodes that are on the paths from the root to the images of the rows of $T_1$ under $h$ (the images are included in this segment).

For each attribute $A \in \bigcup(D)$, we now define the nodes of the finite segment in which $A$ is essential. Let $\text{contain}(A)$ be the set of all the nodes that are images of $h$ and contain $A$, and let node $v$ be the lowest common ancestor of $\text{contain}(A)$. Attribute $A$ is defined to be essential in $v$ and in all the nodes on any path from $v$ leading down to a node in $\text{contain}(A)$. We claim that every node contains its essential attributes. The claim follows from the fact that all the nodes in $\text{contain}(A)$ must have the same variable $b$ in column $A$ (since they are images of rows of $T_1$ having the same variable in column $A$); and, by the method generating the tree, that can happen only if $b$ "trickles" from $v$ (through nodes containing $A$) down to all the nodes in $\text{contain}(A)$. The claim implies the following fact.

**Fact 1:** If two nodes have $A$ as an essential attribute, then there is a path in the finite tree segment that connects them and passes only through nodes in which $A$ is essential. (That path goes from one node up to $v$ and then down to the other node.)

---

**procedure** $DTP(D: \text{schema}, D_T: \text{a tree schema added to } D): \text{boolean};$

begin
  if $D$ is a tree schema then $DTP := \text{true}$
  else if some $R \in D$ divides $D$ into $D_1$ and $D_2$
    then $DTP := DTP(D_1, D_T) \land DTP(D_2, D_T)$
  else if for each $R \in D$ there exists $S \in D_T$ such that $(R \cap M(D)) \subseteq S$
    then $DTP := \text{true}$
    else $DTP := \text{false}$
endor

Figure 2
Let \( D^t \) be a database schema defined as follows. For each node \( v \) in the finite segment, we have a relation schema in \( D^t \) consisting of the attributes that are essential in \( v \). Clearly, \( D^t \) consists of all the \( R_i \) and possibly subsets of the \( R_i \) and the \( S_j \). Consider the finite tree segment as an undirected graph \( G \). We identify each relation schema of \( D^t \) with the node generating it and, thus, the nodes of \( G \) are in one-to-one correspondence with the relation schemas of \( D^t \). By Fact 1, \( G \) is \( \bigcup(D^t) \)-connected and, therefore, is a qual tree of \( D^t \). Thus, \( D^t \) is a TP of \( D \) w.r.t. \( D \). 

4. Solving by Adding a Tree Schema

4.1. Discovering Tree Projections

In this section we consider the problem of determining the existence of a TP \( D^t \) of \( D \) w.r.t. \( D \). The procedure \( DTP \), shown in Figure 2, is a polynomial time algorithm that solves this problem for the case \( D = D \cup D_T \) where \( D_T \) is a tree schema. Observe that adding to \( D \) a single relation schema, or even two relation schemas, is a special case of the above. Let \( M(D) \) denote the attributes which appear in more than one relation schema in \( D \). We say that \( R \in D \) divides \( D \) into database schemas \( D_1 \) and \( D_2 \) if

1. each relation schema of \( D \) is in exactly one of \( D_1 \) and \( D_2 \), except for \( R \) which appears in both,
2. neither \( D_1 \) nor \( D_2 \) has all the relation schemas of \( D \), and
3. \( \bigcup(D_1) \cap \bigcup(D_2) = R \) (actually, there is an equality and not just containment, since \( R \) appears in both \( D_1 \) and \( D_2 \)).

Lemma 1: \( DTP \) is a polynomial time algorithm.

Proof: Let \( size(D) \) denote the length of the string representation of \( D \).

Checking whether \( D \) is a tree schema can be done in \( O(size(D)) \) time [TY].

Checking whether there exists \( R \in D \) which divides \( D \) can be done by checking each \( R \) as follows. (Observe that when this test is performed, \( D \) is not a tree schema and, hence, \( D \) has at least three relation schemas and there is no \( R \in D \) such that \( R = \bigcup(D) \).) Form a graph \( AG(D) \) whose vertices are \( \bigcup(D) \) such that edge \( \{i,j\} \) appears in \( AG(D) \) if attributes \( i \) and \( j \) appear together in some \( R \in D \). This can be done in \( O(size(D)^2) \) which is also the maximum possible size for \( AG(D) \). \( R \) divides \( D \) if removing the nodes (i.e., attributes) of \( R \) from \( AG(D) \) results in a graph with more than one connected component; this can be checked in linear time in the size of \( AG(D) \) by performing a depth first search starting at an arbitrary node. If the search does not mark all the vertices then there are several connected components; \( D_1 \) is formed by all relation schemas containing "marked" attributes (nodes) and \( D_2 \) is formed by all relations containing an "unmarked" attribute. So, the total cost of this check is \( O(n \times size(D)^2) \) where \( D \) has \( n \) relation schemas.
schemata.

If no RE divides D then D is scanned and each occurrence of an attribute is counted; A \in \text{M}(D) if the counter for A is larger than one. This costs \(O(\text{size}(D))\) time. Checking whether for each RE \(D\) there exists SE \(D_S\) such that \((\text{R} \cap \text{M}(D)) \subseteq \text{S}\) can be done as follows. Suppose that \(D_T\) has \(q\) relation schemata and \(t\) attributes; form a \(q \times t\) adjacency matrix whose rows correspond to relation schemata and columns to attributes; this takes \(O(\text{size}(D_T))\) time as matrix initialization can be done in \(O(1)\) (see [AHU]). Now, each relation in D is checked against each matrix row; this costs, in total, \(O(q \times \text{size}(D))\). So, the total cost for this phase is \(O(q \times \text{size}(D) + \text{size}(D_T))\).

It follows that each activation of \(DTP\) takes at most \(O(q \times \text{size}(D) + n \times \text{size}(D)^2 + \text{size}(D_T))\). The number of such recursive calls is given by the following recurrence equation:

\[
T(n) = T(m) + T(n - m + 1) + 2 \\
T(1) = T(2) = 0
\]

In the above equation \(n\) is the number of relation schemata in \(D\), \(m\) is the number of relation schemata in \(D_1\) and \(n - m + 1\) is the number of relation schemata in \(D_2\). The equation states that if \(D\) has \(n\) relation schemata then the total number of calls on \(DTP\) resulting from an initial call is at most 2 (for the calls \(DTP(D_1, D_T)\) and \(DTP(D_2, D_T)\)) plus the number of calls each of these recursive calls will generate. The solution for the above equation is:

\[
T(n) = 2n - 4
\]

Thus, the overall cost of \(DTP\) is bounded by \(O(n \times q \times \text{size}(D) + n^2 \times \text{size}(D)^2 + \text{size}(D_T))\).

Proving the correctness of the algorithm is based on the following three lemmas; in the first two lemmas, \(D_C\) is not necessarily a tree schema.

Lemma 2: Suppose that \(R \in D\) divides \(D\) into \(D_1\) and \(D_2\). Then \(D \cup D_C\) has a TP w.r.t. \(D\) if and only if both \(D_1 \cup D_C\) and \(D_2 \cup D_C\) have TPs w.r.t. \(D_1\) and \(D_2\), respectively.

Proof: If: Suppose that there exists a TP \(D_1^1\) together with a qual tree \(T_1\) of \(D_1 \cup D_C\) w.r.t. \(D_1\), and a TP \(D_2^2\) together with a qual tree \(T_2\) of \(D_2 \cup D_C\) w.r.t. \(D_2\). In both \(T_1\) and \(T_2\) there is a node corresponding to \(R\). We may assume that \(\cup(D_1^1) \subseteq \cup(D_1)\), since attributes not in \(D_1\) may be deleted from \(D_1^1\) and the result is still a TP of \(D_1 \cup D_C\) w.r.t. \(D_1\). Similarly, \(\cup(D_2^2) \subseteq \cup(D_2)\). Form a tree \(T\) by identifying the nodes corresponding to \(R\) in \(T_1\) and \(T_2\). Since \(\cup(D_1) \cap \cup(D_2) \subseteq R\), the tree \(T\) is a qual tree, and the database schema corresponding to \(T\) is a TP of \(D \cup D_C\) w.r.t. \(D\).

Only if. Let \(D_M\) be a tree projection of \(D \cup D_C\) w.r.t. \(D\). Form \(D_1^1\) from \(D_M\) by intersecting each relation schema in \(D_M\) with \(\cup(D_1)\); form \(D_2^2\) from \(D_M\) by intersecting each relation schema in \(D_M\) with \(\cup(D_2)\). Clearly, both \(D_1^1\) and \(D_2^2\) are tree schemata. A schema \(R_2\) of \(D_2\) contributes to \(D_1^1\) only the attributes in \(R_2 \cap (\cup(D_1))\), which is contained in \(R\) (since \(R\) divides \(D_1\) and \(D_2\), and so if all the schemata that originated in \(D_2\) are deleted from \(D_1^1\), the result is a TP of \(D_1 \cup D_C\) w.r.t. \(D_1\).
A TP of $D_2 \cup D_C$ w.r.t. $D_2$ can similarly be obtained. \[ \]

Schema $D^t$ is a minimal TP of $D$ w.r.t. $D$ if there is no TP of $D$ w.r.t. $D$ having fewer relation schemas than $D^t$.

**Lemma 3:** Consider $D \cup D_C$. Suppose that there is no $R \in D$ which divides $D$ into $D_1$ and $D_2$. If $T$ is a qual tree for a minimal TP $D_M$ of $D \cup D_C$ w.r.t. $D$, then all the relation schemas of $D$ are leaves in $T$.

**Proof:** Suppose that an internal node in $T$ corresponds to some $R \in D$. Consider $R$ as the root of $T$. By minimality of $D_M$, each of the subtrees connected to $R$ contains at least one relation schema of $D$. Hence, one can divide $D$ into $D_1$ containing $R$ and the base relation schemas in one subtree, and $D_2$ containing $R$ and the rest of the base relation schemas. By attribute connectivity, $R$ contains all attributes appearing in more than one subtree. Thus $D$ can in fact be divided and we have a contradiction. Therefore, in $T$, no internal node corresponds to a base relation schema. \[ \]

**Lemma 4:** Consider $D \cup D_T$, where $D_T$ is a tree schema. The following are equivalent.

1. For each $R \in D$ there exists $S \in D_T$ such that $(R \cap M(D)) \subseteq S$.

2. There is a TP of $D \cup D_T$ w.r.t. $D$ having a qual tree $T$ in which all the relation schemas of $D$ are leaves.

**Proof:** (1) $\implies$ (2). First, let $D'$ be obtained from $D_T$ by removing from $D_T$ all attributes not in $M(D)$. Consider a qual tree $T'$ for $D'$; attach each relation schema $R \in D$, as a leaf node, to a node corresponding to a relation schema $S \in D_T$ such that $(R \cap M(D)) \subseteq S$. The new tree $T$ is a qual tree for $D \cup D'$.

(2) $\implies$ (1). Since each node corresponding to a base relation is a leaf in $T$, it follows by attribute connectivity that the attributes of each relation schema in $D$, which appear in more than one base relation, must also appear in a projection of some $S \in D_T$. \[ \]

**Theorem 3:** Let $D_T$ be a tree schema. There exists a TP of $D \cup D_T$ w.r.t. $D$ if and only if $DTP(D, D_T)$ returns true.

**Proof:** If. By induction on the number of relation schemas in $D$ we show that there is a TP of $D \cup D_T$ w.r.t. $D$.

**Basis.** If $D$ contains two or fewer relation schemas, then $DTP$ returns true. This is correct because $D$ is a tree schema and hence a TP w.r.t. itself.

\[\]
Induction. Case 1. D is a tree schema. In this case, D is a TP w.r.t. itself.

Case 2. There exists R dividing D into D1 and D2. Since we assume that DTP(D,DT) is true, it follows that so are DTP(D1,DT) and DTP(D2,DT). By the inductive hypothesis, there exists a TP D1 of D1∪DT w.r.t. D1 and a TP D2 of D2∪DT w.r.t. D2. By Lemma 2, there is a TP of D∪DT w.r.t. D.

Case 3. For each R∈D there exists S∈DT such that (R∩M(D)) ⊆ S. By Lemma 4, there is a TP of D∪DT w.r.t. D.

Only if. We will show by induction on the number of relation schemas in D that DTP(D,DT) returns true.

Basis. If D contains two or fewer relation schemas, then D is a tree schema; hence, D is a TP w.r.t. itself and DTP correctly returns true.

Induction. Suppose there exists a TP of D∪DT w.r.t. D. Let T be a qual tree for a minimal such TP D_M.

Case 1. D is a tree schema. In this case, DTP recognizes it and returns true.

Case 2. R∈D divides D into D1 and D2. By Lemma 2, there is a TP D1 of D1∪DT w.r.t. D1, and a TP D2 of D2∪DT w.r.t. D2. By the inductive hypothesis, both DTP(D1,DT) and DTP(D2,DT) return true and, hence, DTP(D,DT) returns true.

Case 3. There is no R∈D which divides D into some D1 and D2. By Lemma 3, all the base relation schemas are leaves in T. By Lemma 4, for each R∈D there exists S∈DT, such that (R∩M(D)) ⊆ S. Therefore, DTP returns true.

The procedure DTP can be modified to construct a TP as follows. Each invocation of DTP returns the TP it has discovered. If D is a tree schema then D itself is returned. Otherwise, if for each R∈D there exists S∈DT such that (R∩M(D)) ⊆ S, then the returned schema is the union of D and D' (where D' is obtained from DT by removing all attributes not in M(D)). In case of recursive calls, the union of the schemas returned by the two calls is the desired result.

Corollary 1: Given a schema D, there is a unique minimal set of attributes Y, such that D∪(Y) has a TP w.r.t. D. The set Y can be found in polynomial time.

Proof: Suppose D_T = (X), i.e. we add a single relation schema to a given database schema D_0. Here, the statement "for each R∈D there exists S∈DT such that (R∩M(D)) ⊆ S" can be simplified to "M(D) ⊆ X." Suppose DTP(D_0,(X)) returns true. Therefore, whenever the test M(D) ⊆ X is performed during the execution of DTP(D_0,(X)), the result is true. Let S be the set of database schemas D for which the test M(D) ⊆ X is performed during the execution of DTP(D_0,(X)), and let
\[ Y = X \cap (U \Delta s M(D)) \] 

Obviously, \( DTP(D_0(Y)) \) would also return \textit{true}. If any attribute is deleted from \( Y \) then some test would be negative and hence \( DTP \) would return \textit{false}. Therefore, \( Y \) is a minimal set of attributes for which \( DTP(D_0(Y)) \) returns \textit{true}. To show that \( Y \) is unique, suppose that there exists a set \( Z_i \), such that \( Y \) is not a subset of \( Z \) and \( DTP(D_0(Z)) \) returns \textit{true}. Consider executing \( DTP(D_0(Z)) \) with the same dividing relations as when \( Y \) is defined. Clearly, some test \( \left( (R \cap M(D)) \subseteq Z \right) \) would be negative and this will cause \( DTP \) to return \textit{false}. (Observe that \( DTP \) does not try all possible divisions, and hence a negative test will cause \textit{false} to be returned for the original call.)

We conclude that \( Y \) is the unique minimal set of attributes for which \( DTP(D_0(Y)) \) returns \textit{true}. If we run the algorithm with \( X = M(D) \), we can compute \( Y \) as defined in (1). \]

Given a database schema \( D \), there is a unique minimal set of attributes \( X \), such that \( D \cup (X) \) is a tree schema [GST]. The unique minimal set \( Y \), such that \( D \cup (Y) \) has a TP w.r.t. \( D \), is contained in but not necessarily equal to \( X \).

4.2. Equivalence for Finite Databases

**Theorem 4:** Let \( \bar{D} = D \cup D_T \), where \( D_T \) is a tree schema. Then the following are equivalent for finite databases:
(1) \( \bar{D} \) solves \( D \).
(2) \( \bar{D} \) strongly reduces \( D \).
(3) \( \bar{D} \) weakly reduces \( D \).
(4) There exists a TP of \( \bar{D} \) w.r.t. \( D \).
(5) There is a monotone join expression of \( D \) over \( \bar{D} \).

**Proof:** The proof of Theorem 2 is valid also for finite databases, except the part showing (2) \( \implies (4) \). So, we show (2) \( \implies (4) \) for finite databases assuming that \( D_T \) is a tree schema. Suppose that the claim (2) \( \implies (4) \) is false. Among all pairs of a database schema \( D \) and a tree schema \( D_T \) serving as counterexamples to the claim, we choose one, denoted \( \bar{D} = D \cup D_T \), such that \( \bar{D} \) has a minimal number of relation schemas among all counterexamples, and \( \cup(D) \) has a minimal number of attributes among all counterexamples with a minimal number of relation schemas. Thus, \( \bar{D} \) is a smallest counterexample to the claim and, so, \( \bar{D} \) strongly reduces \( D \) (when databases are finite), but there is no TP of \( \bar{D} \) w.r.t. \( D \). Under these assumptions we claim the following.

**Claim 1:**
(1) \( R_i \notin R_j \) for all \( R_i, R_j \in D \).
(2) No \( R_i \in D \) has an isolated attribute.

18
Proof: Suppose that there is a pair of relation schemas in \( \mathcal{D} \) such that \( R_i \subseteq R_j \). Let \( \mathcal{D}_1 \) and \( \mathcal{D}_2 \) be obtained by removing \( R_i \) from \( \mathcal{D} \) and \( \mathcal{D} \), respectively. We claim that \( \mathcal{D}_1 \) and \( \mathcal{D}_2 \) form a smaller counterexample, in contradiction to the minimality of \( \mathcal{D} \). In proof, suppose that there is a TP of \( \mathcal{D}_1 \) w.r.t. \( \mathcal{D}_2 \). Since \( R_i \subseteq R_j \), that TP can be made a TP of \( \mathcal{D} \) w.r.t. \( \mathcal{D} \) by adding \( R_i \). Next, let \( \mathcal{D}_1 \) be a state for \( \mathcal{D}_1 \) such that \( \text{SJR}(\mathcal{D}_1) \). We can expand \( \mathcal{D}_1 \) to a state for \( \mathcal{D} \) by assigning to \( R_i \) the relation \( R_i[R_i] \). Since \( \mathcal{D} \) strongly reduces \( \mathcal{D}_1 \), we have \( \text{JR}(R_1, \ldots, R_n) \) and, hence, \( \text{JR}(D_2) \). Therefore, \( \mathcal{D}_1 \) strongly reduces \( \mathcal{D}_2 \).

Now suppose that some \( R_i \in \mathcal{D} \) has an isolated attribute, i.e., an attribute \( A \) that appears only in \( R_i \). Similarly to the proof of the first part of the claim, we can show that the removal of \( A \) from \( \mathcal{D} \) (and hence \( \mathcal{D} \)) creates a smaller counterexample. In particular, note that we can always add to some of the relations in a state for \( \mathcal{D} \) a new attribute that does not already appear in \( \mathcal{D} \), and fill the columns for the new attribute with the constant 1. This expanded state is semijoin reduced if and only if the original state is semijoin reduced. Similarly, when we restrict ourselves to the relations for the relation schemas of \( \mathcal{D} \), then the expanded state is join reduced if and only if the original state is join reduced.

Claim 2: There is no \( R \in \mathcal{D} \) which divides \( \mathcal{D} \) into some \( \mathcal{D}_1 \) and \( \mathcal{D}_2 \).

Proof: Suppose that there is an \( R \in \mathcal{D} \) which divides \( \mathcal{D} \) into \( \mathcal{D}_1 \) and \( \mathcal{D}_2 \). There are three cases to consider, and in each one we derive a contradiction, thereby proving the claim.

Case 1. \( \mathcal{D}_1 \cup \mathcal{D}_T \) has a TP w.r.t. \( \mathcal{D}_1 \) and \( \mathcal{D}_2 \cup \mathcal{D}_T \) has a TP w.r.t. \( \mathcal{D}_2 \). By Lemma 2, there is a TP of \( \mathcal{D} \) w.r.t. \( \mathcal{D} \) in contradiction to our assumptions about \( \mathcal{D} \).

Case 2. \( \mathcal{D}_1 \cup \mathcal{D}_T \) does not have a TP w.r.t. \( \mathcal{D}_1 \). Let \( \mathcal{D}'_T \) be obtained from \( \mathcal{D}_T \) by deleting attributes not in \( \mathcal{D}_1 \). Clearly, \( \mathcal{D}_1 \cup \mathcal{D}'_T \) does not have a TP w.r.t. \( \mathcal{D}_1 \) (since such a TP would also be a TP of \( \mathcal{D}_1 \cup \mathcal{D}_T \) w.r.t. \( \mathcal{D}_1 \)). If \( \mathcal{D}_1 \cup \mathcal{D}'_T \) strongly reduces \( \mathcal{D}_1 \), then \( \mathcal{D} \) is not a smallest counterexample and, so, \( \mathcal{D}_1 \cup \mathcal{D}'_T \) does not strongly reduce \( \mathcal{D}_1 \). We will derive a contradiction by showing that \( \mathcal{D} \) cannot strongly reduce \( \mathcal{D}_1 \). Let \( \mathcal{D}_1, \mathcal{T} = (R_i', \ldots, R_i, S_1, \ldots, S_m) \) be a state of \( \mathcal{D}_1 \cup \mathcal{D}'_T \) showing that \( \mathcal{D}_1 \cup \mathcal{D}'_T \) does not strongly reduce \( \mathcal{D}_1 \), i.e.,

\[
\text{SJR}(D_1, \mathcal{T}) \text{ but not JR}(R_i, \ldots, R_i)
\]  

(A)

We first expand state \( D_1, \mathcal{T} \) to a state for \( \mathcal{D}_1 \cup \mathcal{D}_T \) as follows. The relation schemas \( S_1, \ldots, S_m \) (of the relations \( S_i, \ldots, S_m \) were obtained from those of \( \mathcal{D}_T \) by deleting attributes not in \( \mathcal{D}_1 \). We expand the relations \( S_i, \ldots, S_m \) to a state for \( \mathcal{D}_T \) by adding columns for the deleted attributes and filling these columns with the constant 1. Since the \( R_i \) relations are not changed, the modified state continues to satisfy (A). We now expand this state to a state \( \mathcal{D} \) for \( \mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_T \) by defining a relation for each \( R_i \in \mathcal{D}_2 \). The modified state for \( \mathcal{D}_1 \cup \mathcal{D}_T \) has a relation \( R \) for \( R \). The relation for \( R_i \in \mathcal{D}_2 \) is obtained by adding to the relation \( R[R_i] \) additional columns for the attributes of \( R_i \).
and putting 1 everywhere in these columns. Note that \( R_i - R \) is disjoint from \( \cup(D_i) \), since \( R \) divides \( D \). Therefore, the expanded state \( \overline{D} \) satisfies \( \text{SJR}(\overline{D}) \) but not \( \text{JR}(R_1, \ldots, R_n) \). Thus, this state contradicts the assumption that \( D \) strongly reduces \( D \).

**Case 3.** \( D_2 \cup D_7 \) does not have a TP w.r.t. \( D_2 \). This case is similar to Case 2.

**Claim 3:** For every \( R \in D \) there is an \( S \in \overline{D} \), such that \( R \subseteq S \).

**Proof:** Suppose that some \( R \in D \), say \( R_1 \), is not contained in any \( S \in \overline{D} \). By Claim 1, \( R_1 \) is also not contained in any other \( R_i \). Consider the expressions \( E_1 \) and \( E_2 \) defined in the proof of Theorem 2. We now perform a modified chase w.r.t. \( I \) (the set of IDs defined in Section 3.4) on the tableau \( T_2 \) corresponding to \( E_2 \). We choose a variable \( d \), initially not in \( T_2 \), and use \( d \) each time a new variable has to be introduced according to the ordinary chase rules. Thus, \( d \) is the only new variable in \( \text{chase}(T_2) \), and so \( \text{chase}(T_2) \) is finite.

**Case 1.** There is no containment mapping from \( T_1 \) into \( \text{chase}(T_2) \). Therefore, \( \text{chase}(T_2) \) is a finite state \( \overline{D} \), such that \( \text{SJR}(\overline{D}) \) but not \( \text{JR}(R_1, \ldots, R_n) \), and so it shows that \( D \) does not strongly reduce \( D \); a contradiction.

**Case 2.** There is a containment mapping \( h : T_1 \rightarrow \text{chase}(T_2) \). Since \( R_1 \) is not contained in any other relation schema of \( D \), only the root of \( \text{chase}(T_2) \) has distinguished variables in all the columns of \( R_1 \), and so \( h \) maps row \( R_1 \) of \( T_1 \) to this root. If \( h \) maps all the other rows of \( T_1 \) to the descendants of a single child of the root, then there is some \( A \in R_1 \) that does not appear in any other \( R_i \) in contradiction to Claim 1. If \( h \) maps the rows of \( T_1 \) (other than \( R_1 \)) to descendants of at least two children of the root, then \( R_1 \) divides \( D \) into \( D_1 \) and \( D_2 \), where \( D_1 \) consists of \( R_1 \) and the relation schemas that are mapped to the descendants of one child of the root, and \( D_2 \) has \( R_1 \) and all the other descendants \( R_i \). But this is a contradiction to Claim 2.

Since all the possible cases lead to a contradiction, the claim is proved.

Claim 3 and Lemma 4 imply that \( \overline{D} \) has a TP w.r.t. \( D \), in contrary to the assumption that \( \overline{D} \) is a counterexample to (2) \( \Rightarrow \) (4). Therefore, (2) \( \Rightarrow \) (4). \( \square \)

### 5. Conclusions

Given a schema \( D \) and a state \( D \) for \( D \), the join of all relations in \( D \) projected on the attributes of some \( R \) in \( D \) is desired. One way of computing the desired result is enlarging \( D \) into \( \overline{D} \) and \( D \) into \( \overline{D} \), using the join and project operators, and then operating on \( \overline{D} \) by deleting tuples until the desired result is obtained. A method of performing such a process is reduction, i.e. transforming \( \overline{D} \) into a maximal semi-join reduced state.

A basic problem of relational query processing is that of characterizing the structure of \( D \). We have formalized this problem and solved it for the case in which finite as well as unrestricted
database states are allowed. We have also solved the problem for the case in which \( D \) is formed by adding a tree schema to \( D \) and only finite database states are allowed. In both cases, we showed that \( D \) must include a tree projection with respect to \( D \). In the latter case, i.e. adding a tree schema to \( D \), we designed a polynomial time algorithm which, when given \( D \) and \( D \) thus formed, decides whether there exists a tree projection of \( D \) with respect to \( D \).

Two interesting open problems remain. One is that of discovering whether, in the case of finite databases, a tree projection is indeed necessary. The other is that of designing a polynomial time algorithm for deciding tree projection existence over arbitrary schemas, or showing that the problem is NP-complete.

References


