LOCKING EXPRESSIONS
FOR INCREASED DATABASE CONCURRENCY

by

Anthony Klug

Computer Sciences Technical Report #400

October 1980

Locking Expressions for Increased Database Concurrency

Anthony Klug

University of Wisconsin

Abstract

Access to a relation R in a relational database is sometimes based on how R joins with other relations rather than on what values appear in the domains of R. Using simple predicate locks forces the entire relation to be locked in these cases. In this paper a technique is presented which allows locking of the smallest possible set of tuples even when the selection is based on joins with other relations. The algorithms are based on a generalization of tableaux. The tableaux used here can represent relational algebra queries with the entire set of domain comparison operators '=', '$\neq$', '<', '$\leq$', '>', '$\geq$'.

Keywords and Phrases: predicate lock, concurrency control, relational database, relational algebra, expression lock, tableau, transaction, well-formed, two-phased, legal history

CR Categories: 4.32, 4.33

Author's address: Computer Sciences Dept., University of Wisconsin-Madison, Madison, WI 53706

Contents

1 Introduction 1

1.1 Overview of Paper 4

2 Relational Definitions 4

3 Transactions and Schedules 7

4 Tableaux 18

5 Tableau Based Lock Algorithms 31

6 Extensions for the Update Operation 33

7 An Example 36

8 Summary and Conclusions 37

9 Appendix 38

1. Introduction

In any database management system allowing simultaneous access and modification by several users, a method must be provided for synchronizing these actions. Several approaches to this concurrency control problem are possible [BeGo2]. Locking is perhaps the most familiar.

Locking controls concurrency by limiting access to entities in the database by declaring them "locked". In the relational model, the natural unit of access, i.e. the locking entity, is a set of tuples. Since mathematical sets are specified by properties or formulas, it follows that locks on relations may be specified by properties or formulas. In the past, these properties were limited to referring only to the attributes of the relation being locked, and these were called "simple predicate locks" [EGLT], [WoEd]. In this paper, we generalize the properties allowed in locks, and we call them "expression locks".

To motivate the introduction of expression locks, we present a simple example.

Consider the relational schema of Figure 1. Suppose the following two transactions are run against this schema:

TR_1: Give employees whose hire date is less than 780630 (hired before June 30, 1978) a 10% raise.

```
dept(dno, dname, manager, budget)
  key(dno)
employee(eno, ename, sal, hiredate, edno)
  key(eno)
  foreign key(dno)
```

Figure 1. Schema for Departments and Employees.

TR_2: Give employees whose hire date is greater than 800901 (hired after Sept. 1, 1980) a 5% pay cut.

The sets of employee tuples accessed by TR_1 and TR_2 can be defined with simple predicates[1]:

$$E_1 = \{e \in \text{employee} : \text{e.hiredate} < 780630\}, \text{ and}$$
$$E_2 = \{e \in \text{employee} : \text{e.hiredate} > 800901\}.$$

It is clear that these two sets are disjoint since the conditions (e.hiredate < 780630) and (e.hiredate > 800901) cannot both be satisfied by the same tuple. Thus TR_1 and TR_2 can be run concurrently without affecting the integrity of the database.

Next consider the following pair of transactions:

TR_3: Give every employee in departments with a budget exceeding $1 million a 10% raise.

TR_4: Give every employee in departments with a budget below $500,000 a 5% pay cut.

Intuitively, we can see that this pair of transactions also can be run concurrently since the set of all employees in "low budget" departments is disjoint from the set of all employees in "high budget" departments. (Note that this reasoning makes implicit use of the fact that dno is the key of the dept relation.) For this pair of transactions, the sets of employee tuples accessed are not defined by simple predicates:

$$E_3 = \{e \in \text{employee} : (\exists d \in \text{dept})(\text{d.dno=e.dno} \ \& \ \text{d.budget}>1000000\} \text{ and}$$
$$E_4 = \{e \in \text{employee} : (\exists d \in \text{dept})(\text{d.dno=e.dno} \ \& \ \text{d.budget}<900000\}$$

[1] A simple predicate as defined in [EGLT] is a boolean combination of terms of the form (a_1 θ a_2) or (a_1 θ c), where a_1 and a_2 are attributes of a single relation, θ is '=', '<', etc., and c is a constant.

In order to lock these two sets, it is suggested in [EGLT] that two simple predicates be found which "cover" the two defining formulas. In this case, there are no restriction or selection terms on employee attributes, and so the smallest covering simple predicate in both cases selects the entire employee relation. Hence if we use only simple predicates, transactions TR_3 and TR_4 must be run serially.

Now let us consider how we might remedy this situation. Proving that the two sets E_3 and E_4 are disjoint as long as the key constraint for the dept relation are satisfied, is equivalent to proving that the two defining formulas are not mutually satisfiable as long as the key of the dept relation is satisfied. Thus we must prove the inconsistency of the formula:

$$
\begin{array}{l}
(\forall d_1,d_2 \in \text{dept})(d_1.\text{dno}=d_2.\text{dno} \Rightarrow d_1.\text{budget}=d_2.\text{budget}) \ \& \\
(\exists e \in \text{employee})(\\
(\exists d \in \text{dept})(d.\text{dno}=e.\text{dno} \ \& \ d.\text{budget}>1000000 \ \& \\
(\exists d' \in \text{dept})(d'.\text{dno}=e.\text{dno} \ \& \ d'.\text{budget}<500000))
\end{array}
$$

This formula happens to be equivalent to a π_2-formula[2], and so there is a decision procedure [Acke] for determining its satisfiability. Unfortunately, the algorithm is superexponential in the number of universally quantified variables (which is actually eight in the above example, one for each attribute of the two dept tuples), and it seems desirable to look elsewhere for a more efficient and intuitive solution. In this paper, we study the use of tableaux for this problem.

Tableaux have been used by a number of authors to solve important problems in the relational database model. For example, they have been used to optimize relational expressions [ASU1], to test dependency statements [MaMS] and to check correctness of views [KlPr]. The

[2] Put in prenex normal form, all universal quantifiers precede all existential quantifiers.

tableaux used in these works can only represent relational expressions which contain only equality comparisons among domains and between domains and constants. We need tableaux which can model queries with the comparison operators "less-than", "greater-than-or-equal", etc. Such tableaux could be used in the above locking problem. In this paper we introduce the notion of "inequality tableaux", tabular representations of relational algebra expressions which do just this. We prove some basic properties of inequality tableaux and then we then use them for representing "expression locks".

1.1. Overview of Paper

Our goal is to give algorithms for locking which can be used to allow concurrent execution of transactions such as TR_3 and TR_4 given above. Before we do this we present the necessary relational terminology (Section 2). Then in Section 3 we define our notions of expression locks, well-formed two-phased transactions, and legal histories. The main theorem is that legal histories of well-formed two-phased transactions are serializable. In Section 4 we introduce the notion of inequality tableau, and we prove some useful theorems for them. Then in Section 5 we show how the tableau algorithm can be applied for our expression locking scheme. We close with a discussion of the update operation and an example.

2. Relational Definitions

The formal model we use does not make the universal instance assumption. A <u>relation scheme</u> is a pair <R,k>. R is a symbol (the relation name), and k is a positive integer (R's degree) which is denoted deg(R). If <R,k> is a relation scheme, the domains of R, doms(R), is the set {1,2,...,k} of natural numbers. A <u>functional</u>

<u>dependency</u> (<u>FD</u>) is a triple $\langle R_i, Z, A\rangle$, also written $R_i{:}Z \to A$, where R is a relation of degree k, $Z \subseteq \{1,2,\ldots,k\}$, $A \in \{1,2,\ldots,k\}$ and $A \notin Z$. A <u>schema</u> is a pair $\langle S,C\rangle$, where S is a sequence $\langle\langle R_1,k_1\rangle,\ldots,\langle R_N,k_N\rangle\rangle$ of relation schemes which is sometimes written simply $\langle R_1,\ldots,R_N\rangle$, and where C is a set of FDs on the relations given. Throughout this paper, one fixed schema $\langle R_1,\ldots,R_N\rangle$ is assumed. An <u>instance</u> I of schema $\langle\langle R_1,\ldots,R_N\rangle,C\rangle$ is an N+2-tuple $\langle D,O,I_1,\ldots,I_N\rangle$, where D is the <u>domain</u> <u>of</u> <u>values</u>, O is a partial, asymmetric, transitive order[3], and for each $i=1,\ldots,N$, $I_i \subseteq D^{\deg(R_i)}$. Domains of all relations are taken without loss of generality to range over the set D, and D^m is the set of all m-tuples over D. For convenience, we will assume that the set $\mathbb{N}$ of natural numbers is embedded in D and that the less-than relation on natural numbers is consistent with the order on D[4]. We denote the class of all instances by $\mathbb{I}$. An FD $R_i{:}Z \to A$ is <u>true</u> in instance I if for all tuples t_1,t_2 in I_i, if $t_1[Z]=t_2[Z]$, then $t_1[A]=t_2[A]$. Brackets '[', ']' denote project on the listed domains. A <u>state</u> S of schema $\langle\langle R_1,\ldots,R_N\rangle,C\rangle$ is an instance in which all constraints in C are true.

The set $\mathbb{E}$ of <u>expressions</u> over our fixed schema and the associated functions <u>deg</u> (degree) and <u>doms</u> (domains) for expressions are defined as follows:

If $e \in \mathbb{E}$ has degree k, then $doms(e) = \{1,\ldots,k\}$.

(1) Base Relations: $R_i \in \mathbb{E}$ for each R_i in the schema, and $\deg(R_i)$ is

---

[3] For all x,y, if xBy then not yBx, and for all x,y,z, if xBy and yBz, then xBz. For example, the '<' relation on numbers is asymmetric and transitive (and total).

[4] What we really want is the kind of instances (interpretations) which appear in mathematical logic in which the formal languages have constant symbols and the interpretations have interpretations of constant symbols. To simplify matters for those not familiar with models in logic, we use the natural numbers both as constant symbols in our language (relational algebra) and as their own interpretations in the instances.

already defined.

(2) Literals: If $c \in \mathbb{N}$, then $\{c\} \in \mathbb{E}$, and $\deg(\{c\}) = 1$.

(3) Projection: If $e \in \mathbb{E}$, then $e[X] \in \mathbb{E}$ where X is a sublist of doms(e), and $\deg(e[X])$ = length of X.

(4) Cross Product: If $e_1, e_2 \in \mathbb{E}$ and $\deg(e_1) = d_1$, $\deg(e_2) = d_2$, then $(e_1 \times e_2) \in \mathbb{E}$, and $\deg(e_1 \times e_2) = d_1 + d_2$.

(5) Restriction: If $e \in \mathbb{E}$, $X,Y \in$ doms(e) and θ is '=' or '<', then $e[X\theta Y] \in \mathbb{E}$ and $\deg(e[X\theta Y]) = \deg(e)$.

(5) Union: If $e_1, e_2 \in \mathbb{E}$ and $\deg(e_1) = \deg(e_2)$, then $(e_1 \cup e_2) \in \mathbb{E}$, and $\deg(e_1 \cup e_2) = \deg(e_1)$.

With these operators we can also define selections, joins and intersections:

Selection: $e[X\theta V]$ is $(e \times \{V\})[X\theta X'][1,\ldots,\deg(e)]$
where X' is $\deg(e)+1$.

Join: $e_1[X\theta Y]e_2$ is $(e_1 \times e_2)[X\theta Y']$
where Y' is $\deg(e_1)+Y$.

Intersection: $e_1 \sqcap e_2$ is $(e_1[D_1=D_2]e_2)[D_1]$
where D_1 is $<1,\ldots,\deg(e_1)>$ and
D_2 is $<1,\ldots,\deg(e_2)>$

The operators '$\leq$' and '$\neq$' can be defined in terms of '=', '<' and union. Generalized restrictions $e[\Psi]$, where Ψ is a boolean combination of selections and restrictions, can also be defined using repeated restrictions, selections and unions. In certain cases we will also allow the Set Difference operator, '-'. Its formation rules are the same as for union, and the set of expressions including set difference will be denoted $\mathbb{E}^-$. Unless stated otherwise, by "expression" we mean an element of $\mathbb{E}$.

For each $e \in \mathbb{E}^-$ of degree k and for each $I \in \mathbb{I}$, the value of e on I, denoted $e(I)$, is a subset of D^k. The formal definition, which is omitted, gives the usual semantics for relational algebra operators.

An expression e is universally empty (u.e.), written $e \equiv \emptyset$, if $e(S) = \emptyset$ for all states S.

We wish to consider two concepts of "contained in" for expressions. We will write $e_1 \subseteq e_2$, if $e_1(I) \subseteq e_2(I)$ for all instances I. We will write $e_1 \ll e_2$ if $e_1(S) \subseteq e_2(S)$ for all states S. If the set C of schema FDs is empty, '$\subseteq$' is the same as '$\ll$'.

An operation is a statement of the form

insert R e, or
delete R e.

Here, R is a schema relation, and e is an expression of the same degree as R. We give these operations semantics by considering them to be functions on instances with values given by the rules:

If $I' = (\text{insert } R_i \ e)(I)$, then

$$I'_i = I_i \cup e(I) \text{ and } I'_j = I_j, \text{ for } j \neq i.$$

If $I' = (\text{delete } R_i \ e)(I)$, then

$$I'_i = I_i - e(I) \text{ and } I'_j = I_j, \text{ for } j \neq i.$$

3. Transactions and Schedules

In Section 1 we saw how simple predicate locks would not allow some kinds of transactions to be run concurrently even though this was theoretically possible. In this section we introduce "expression locks". This is a generalization of the notion of predicate lock. We develop corresponding concepts of transactions, history, well-formed transactions and legal histories using the notion of expression locks. The main result of the section is the theorem that legal histories of two-phased well-formed transactions are serializable (preserve consistency).

An expression lock, or simply, lock, is a statement of the form:

lock M R e

Here, M is the mode of the lock (S = share, X = exclusive); R is a schema relation, and e is an expression of the same degree as R. (We will often use 'M' to denote either 'S' or 'X'.) Intuitively, a lock statement requests a lock on the set of R-tuples which do or might appear in e.

We also have corresponding unlock statements of the form:

unlock M R e

Intuitively, an unlock statement says to release the R-tuples which do or might appear in e.

A transaction is a (finite) sequence of operations, lock statements and unlock statements. A transaction TR = $<s_1,\ldots,s_m>$ can be considered a function on instances by defining:

$$TR(I) = s_m(\ldots s_2(s_1(I)) \ldots)$$

where $s_i(I) = I$ if s_i is a lock or an unlock statement. We assume that all statements in a transaction preserve the FDs in the schema. Although some treatments on locks allow a transaction to violate a schema constraint temporarily, (for example, a transaction which transfers money from one account to another) this is not acceptable for functional dependencies. Hence, we assume that every step of a transaction preserves the schema FDs. In other words, if S is a state, then $s_i(S)$ is a state.

Sets of tuples, like any mathematical set, behave differently as locking entities than objects which are indivisible. If a set A of tuples is locked, it makes perfect sense to unlock some proper subset B of A, still leaving A-B locked. Thus we should not require of a transaction TR, that if TR accesses a set A of tuples at step i, then TR must have some lock preceding step i which explicitly locks exactly set A. It is really only necessary that the aggregate of all locks (minus all unlocks) includes A. Thus to record the locks and unlocks which have appeared in a transaction, we define the following function:

$$\text{lock} : \mathbb{TR} \times \{S,X\} \times \mathbb{N} \times \{R_1,\ldots,R_N\} \rightarrow \mathbb{E}^-$$

Here, **TR** is the set of all transactions; {S,X} are the locking modes (share, exclusive); the third argument is the index of a statement in the first argument, and the last argument is the set of schema relations. The values of this function are defined as follows, where we assume TR has the form $\langle s_1,\ldots,s_m\rangle$:

$\text{lock}(TR,S,\emptyset,R_j) = \emptyset$ for $j=1,\ldots,N$

$\text{lock}(TR,X,\emptyset,R_j) = \emptyset$ for $j=1,\ldots,N$

If s_i is "lock M R_i e" (M = S or X)

$$\text{lock}(TR,M,i,R_j) = \text{lock}(TR,M,i-1,R_j) \cup e$$

If s_i is "unlock M R_j e"

$$\text{lock}(TR,M,i,R_j) = \text{lock}(TR,M,i-1,R_j) - e$$

Otherwise

$$\text{lock}(TR,M,i,R_j) = \text{lock}(TR,M,i-1,R_j)$$

Next, we want to specify when a transaction is well-formed. Intuitively, any part of a relation which is read, i.e., which is used in the second operand of an operation, must be locked in S mode[5], and any part of a relation which will be updated must be locked in X mode. That is, to execute an operation "op R e" on I, TR must first "read" e(I) and then modify R. To read e(I), a set r_i of tuples is read from I_i (i=1,...,N). These are the tuples that "participate" in forming e(I). The sequence $\langle r_1,\ldots,r_N\rangle$ has the property that the value of e on I is the same as the value of e on $\langle D,O,r_1,\ldots,r_N\rangle$ since each tuple of each r_i "participates" in forming the value of e, and only these tuples participate. This concept is similar to the semi-join of [BeGo1], but here we are not restricted to how we get read sets. The read set of an expression $e \in \mathbb{E}$ can be obtained as follows:

First assume e contains no unions. We may then write e as an ordered cross product followed by some restrictions and selections followed by a projection [Ullm]:

$$(R_1\times \cdots \times R_1 \times \cdots \times R_N \times \cdots \times R_N)[\sigma][Z]$$

Here, each R_i appears zero or more times, and σ represents the conjunction of all restrictions and selections. For each i=1,...,N, we let RS_i be the union over every occurrence of R_i of all expressions of the form:

$$(R_1\times \cdots \times R_1 \times \cdots \times R_N \times \cdots \times R_N)[\sigma][D_{ik}]$$

where D_{ik} projects on the k-th occurrence of R_i. The
read set of e is then the N-tuple of expressions $\langle RS_1,\ldots,RS_N\rangle$.

If e contains unions, it may be written as:

[5] It is possible to generalize this condition to be "in S or X mode" if corresponding changes are made to subsequent definitions and theorems.

$$e_1 \cup \cdots \cup e_n$$

where each e_i contains no unions. The i-th component of the read set of e is then

$$RS_{i1} \cup \cdots \cup RS_{in},$$

where RS_{ij} is the i-th component of the read set of e_j. Read sets have three important properties stated in the following theorem. Its proof is delayed until we have developed tableaux.

<u>Theorem 1</u>. Let e be an expression; let $RS = \langle RS_1,\ldots,RS_N \rangle$ be its read set.

(1) For all instances I, $RS_i(I) \subseteq I_i$ $(i=1,\ldots,N)$ and $e(I) = e(RS(I))$. That is, applying the read set does not change the value of e.

(2) If e is optimized [ASU1] and if $F = \langle F_1,\ldots,F_N \rangle$ is an N-tuple of expressions with the properties that for all I, $F_i(I) \subseteq I_i$ $(i=1,\ldots,N)$ and $e(I) = e(F(I))$, then $RS_i \subseteq F_i$, $i=1,\ldots,N$. That is, the read set is the smallest set of expressions with property (1).

(3) Let e' be an expression of degree $\deg(R_j)$. For any instance I, if $RS_j(I) \cap e'(I) = \emptyset$, then for each $i=1,\ldots,N$, $RS_i(I') = RS_i(I")$ $= RS_i(I)$, where I' and I" are defined by $I'_j = I_j \cup e'(I)$ and $I"_j$ $= I_j - e'(I)$ and $I'_k = I"_k = I_k$ for $k \neq j$.

In property (3), the case for set difference can be derived from properties (1) and (2). The case for union can be rephrased as follows: Since we always have $RS_i(I) \subseteq I_i$, the condition on e' can be written $RS_i(I) \cap (I_i \cap e'(I)) = \emptyset$. Thus when the part of $e'(I)$ in I_i does not intersect $RS_i(I)$, none of the tuples in $(e'(I) - I_i)$ (the part of

$e'(I)$ not in I_i) will join with any tuple in the read set of e. This property can be compared with the property of unions contained in other unions found in [SaYa].

We therefore define a transaction TR = $<s_1, s_2, \ldots, s_m>$ to be <u>well-formed</u> if for each step s_i, if s_i is "op R_j e", then the following two conditions hold:

(1) If the read set of e is $<RS_1, \ldots, RS_N>$, then for each $k=1,\ldots,N$, $RS_k \ll \text{lock}(TR,S,i,R_k)$.

(2) $e \ll \text{lock}(TR,X,i,R_j)$.

A transaction $<s_1, \ldots, s_m>$ is <u>two-phased</u> if there is some step s_i such that s_i is an unlock statement, and for no j, $m \geq j > i$, is s_j a lock statement.

A <u>history</u> h for transactions $TR_1, \ldots, TR_n$ is a sequence of statements such that for each i, every statement of TR_i appears in h exactly once along with the index i of TR_i. (Formally, h is a sequence of pairs $<i,s>$.), and if s_k precedes s_j in TR_i, then $<i,s_k>$ precedes $<i,s_j>$ in h.

A history h = $<s_1, \ldots, s_n>$ defines a function on instances by the composition of the functions associated with the operations in h.

We next a "Lock" function which is analogous to the lock function for transactions (It has one additional argument for the history):

$Lock(h,TR_k,M,\emptyset,R_j) = \emptyset$ for $j=1,\ldots,N$

If s_i is <k, lock M R_i e>

$$Lock(h,TR_k,M,i,R_j) = Lock(h,TR_k,M,i-1,R_j) \cup e$$

If s_i is <k, unlock M R_j e>

$$Lock(h,TR_k,M,i,R_j) = Lock(h,TR_k,M,i-1,R_j) - e$$

Otherwise

$$Lock(h,TR_k,M,i,R_j) = Lock(h,TR_k,M,i-1,R_j)$$

A history h for $TR_1,\ldots,TR_n$ is <u>serial</u> if there is some permutation $\langle p_1,\ldots,p_n\rangle$ of $\{1,\ldots,n\}$ such all statements of TR_{p_i} appear in h before those of $TR_{p_{i+1}}$, for $=1,\ldots,k-1$. Two histories h_1, h_2 are <u>equivalent</u> if they have the same value on every state. A history h is <u>serializable</u> if it is equivalent to a serial history. It is the serializable histories which are "correct" [Papa], [BeGo2]. Although histories always preserve the schema FDs, unserializable histories may violate other database constraints (for example, the sum of one set of attributes equals the sum of another).

A history $h = \langle s_1,\ldots,s_m\rangle$ for transactions $TR_1,\ldots,TR_n$ is <u>legal</u> if for each $k,k'=1,\ldots,n$, with $k\neq k'$, each $i=1,\ldots,m$ and each $j=1,\ldots,N$, we have:

$$Lock(h,TR_k,X,i,R_j) \sqcap$$
$$(Lock(h,TR_{k'},S,i,R_j) \cup Lock(h,TR_{k'},X,i,R_j)) \equiv \emptyset$$

Legal histories can also be determined by the following equivalent property:

For every lock statement <k, lock M R_j e> (at step i) in h and every $TR_{k'}\neq TR_k$:

(1) If $M = S$, then $e \sqcap \mathrm{Lock}(h, TR_{k'}, X, i, R_j) \equiv \emptyset$.

(2) If $M = X$, then $e \sqcap \mathrm{Lock}(h, TR_{k'}, X, i, R_j) \equiv \emptyset$ and $e \sqcap \mathrm{Lock}(h, TR_{k'}, S, i, R_j) \equiv \emptyset$.

The main theorem of this section shows that our definitions have the desired properties:

<u>Theorem 2</u>. Legal histories of two-phased well-formed transactions are serializable.

Proof. Let h be a history for $TR_1, \ldots, TR_n$. We assume that the first unlock in h belongs to TR_1. We will show that h is equivalent to the history $TR_1 h'$, where h' is obtained from h by deleting all of TR_1's statements. The theorem will then follow by induction on the number of transactions. To show that we can move TR_1's statements to the front, it is sufficient to show that if a portion of h has the form:

$$\cdots \langle p,\ stmt_p\rangle \quad \underset{\text{-step m-}}{\langle 1,\ stmt_1\rangle} \cdots$$

then the history obtained by exchanging these two statements is equivalent to the original. We consider the following cases:

(1) TR_p locks or unlocks R_j and TR_1 refers to a different relation R_i:

$$\cdots \langle k,\ \text{un/lock } M\ R_j\ e'\rangle \ \langle 1,\ \text{op/un/lock } M\ R_i\ e\rangle \cdots$$

(2) TR_1 locks or unlocks R_i and TR_p refers to a different relation R_j:

$$\cdots \langle k,\ \text{op/un/lock } R_j\ e'\rangle \ \langle 1,\ \text{un/lock } M\ R_i\ e\rangle \cdots$$

(3) Both TR_p and TR_1 put locks on the same relation:

$$\cdots \langle k,\ \text{lock } M\ R_i\ e'\rangle \ \langle 1,\ \text{lock } M\ R_i\ e\rangle \cdots$$

(4) TR_p locks part of R_i, and TR_1 unlocks part of R_i:

$$\cdots \langle k, \text{lock } M \ R_i \ e' \rangle \langle l, \text{unlock } M \ R_i \ e \rangle \cdots$$

(5) TR_p unlocks part of R_i, and TR_l locks part of R_i:

$$\cdots \langle k, \text{unlock } M \ R_i \ e' \rangle \langle l, \text{lock } M \ R_i \ e \rangle \cdots$$

(6) TR_p unlocks part of R_i, and TR_l unlocks part of R_i:

$$\cdots \langle k, \text{unlock } M \ R_i \ e' \rangle \langle l, \text{unlock } M \ R_i \ e \rangle \cdots$$

(7) TR_p locks or unlocks part of R_i, and TR_l does an operation on R_i:

$$\cdots \langle k, \text{un/lock } M \ R_i \ e' \rangle \langle l, \text{op } R_i \ e \rangle \cdots$$

(8) TR_p does an operation on R_i, and TR_l locks or unlocks part of R_i:

$$\cdots \langle k, \text{op}' \ R_i \ e' \rangle \langle l, \text{un/lock } M \ R_i \ e \rangle \cdots$$

(9) Both TR_p and TR_l do operations (on not necessarily the same relation):

$$\cdots \langle k, \text{op}' \ R_j \ e' \rangle \langle l, \text{op } R_i \ e \rangle \cdots$$

When at most one operation is involved, we must show that the result of reversing the two statements will still be a legal history since it is clear that the function defined by the history will not change. When two operations are involved, we must show that they do not "interfere" with each other.

The first eight cases will be argued intuitively:

Case 1: The lock or unlock of TR_p will still be legal when its position is switched since the lock expressions for R_j will not change.

Case 2: The argument is the same as in case 1.

Case 3: Since the empty intersection property holds after both locks are made, it will also hold after any one of them is made.

Case 4: If we do TR_l's unlock before TR_p's lock, we cannot violate the empty intersection property of the history since the lock expressions when TR_p requests its lock will only be smaller.

Case 5: This case cannot occur since TR_1 has the first unlock in h and because TR_1 is two-phased.

Case 6: Unlocks can only make lock expressions smaller. The empty intersection property cannot be violated by reversing the order of two unlock statements.

Case 7: The effect of the operation cannot change, and the lock or unlock of TR_p will still be legal since the operation of TR_1 does not change lock expressions.

Case 8: The argument is the same as in case 7.

Case 9: Let $I_\emptyset$ be the result of $\langle s_1,\ldots,s_{m-2}\rangle$ on instance I. Define:

$$I_1 = (op'\ R_j\ e')(I_\emptyset)$$
$$I_2 = (op\ R_i\ e)(I_\emptyset)$$
$$I_3 = (op'\ R_j\ e')(I_2)$$
$$I_4 = (op\ R_i\ e)(I_1)$$

What we want to show is that $I_3 = I_4$. We first show that $e(I_\emptyset) = e(I_1)$.

Let RS be the read set of e. It is sufficient to show that $RS(I_\emptyset) = RS(I_1)$. Because we have:

$$e' \ll Lock(h,TR_p,X,m,R_j),$$
$$RS_j \ll Lock(h,TR_1,S,m,R_j) \quad \text{and}$$
$$Lock(h,TR_p,X,m,R_j) \sqcap Lock(h,TR_1,S,m,R_j) \equiv \emptyset,$$

we have $RS_j \sqcap e' \equiv \emptyset$. By property (3) of Theorem 1, we then have $RS(I_\emptyset) = RS(I_1)$, and so $e(I_\emptyset) = e(I_1)$.

By an analogous argument, we can show that $e'(I_\emptyset) = e'(I_2)$.

Now if $i \neq j$, we have the following, where '$\pm$' denotes either set union or set difference, depending on which particular operations op and op' are:

$$
\begin{aligned}
I_{3j} &= I_{2j} \pm e'(I_2) \\
&= I_{\emptyset j} \pm e'(I_2) \\
&= I_{\emptyset j} \pm e'(I_\emptyset) \\
&= I_{1j} \\
&= I_{4j}, \text{ and} \\
I_{4i} &= I_{1i} \pm e(I_1) \\
&= I_{\emptyset i} \pm e(I_1) \\
&= I_{\emptyset i} \ e(I_\emptyset) \\
&= I_{2i} \\
&= I_{3i}
\end{aligned}
$$

Other components have not changed, hence $I_3 = I_4$.

If $i = j$, we have $\mathrm{Lock}(h,TR_p,X,m,i) \sqcap \mathrm{Lock}(h,TR_1,X,m,i) = \emptyset$. Therefore, $e(I_\emptyset) \sqcap e'(I_\emptyset) = \emptyset$, and we have:

$$
\begin{aligned}
I_{3i} &= I_{2i} \pm e'(I_2) \\
&= (I_{\emptyset i} \pm e(I_\emptyset)) \pm e'(I_2) \\
&= (I_{\emptyset i} \pm e(I_\emptyset)) \pm e'(I_\emptyset) \\
&= (I_{\emptyset i} \pm e'(I_\emptyset)) \pm e(I_\emptyset) \\
&= (I_{\emptyset i} \pm e'(I_\emptyset)) \pm e(I_1) \\
&= I_{1i} \pm e(I_1) \\
&= I_{4i}.
\end{aligned}
$$

Hence we also have in this case $I_3 = I_4$.

□

4. Tableaux

In the last section we defined a general framework for a lock-based concurrency control which we called expression locking. We guarantee consistency of the database by ensuring that transactions are well-formed and two-phased and that histories are legal. The conditions for well-formed transactions and for legal histories require that we be able to determine when an expression is universally empty and when one expression is contained in another. In this section we develop algorithms for these problems using the tableau technique.

Tableaux (e.g., [KlPr], [ASU2], [ChMe], [SaYa]) are a shorthand notation for relational expressions. Previous definitions of tableaux have modeled only projections, equi-selections and natural joins on universal instances [ASU2] or projections, equi-selection and equi-restriction and cross product [KlPr] [ChMe] on arbitrary instances. Here however, we need a more general concept of tableau which can represent relational algebra operators in which restrictions may have the "less-than" operator. To motivate the definition, we start by considering the conjunctive queries of Chandra and Merlin. (For more details on the tableaux presented here, see [Klug].)

A <u>conjunctive query</u> is a first-order predicate calculus formula of the form:

$$(x_1,\ldots,x_k).\exists x_{k+1}\ldots x_m.A_1\&\ldots\&A_r,$$

where each A_i is an atomic formula $R_j(t_1,\ldots,t_p)$, where each term t is a variable or a constant. The tableau for such a query is obtained by collecting, for each relation R_j, the arguments of each atomic formula for R_j into a table. We could generalize the set of formulas considered conjunctive queries by extending the allowable atomic formulas: Allow atomic formulas of the form $(t_1 < t_2)$, again where the t's are variables or constants. We can collect the old atomic formulas into tables as before, and the new ones we can collect into a boolean matrix which has a row and a column for each variable and constant appearing in the query. The formal definitions follow. (See the appendix for a discussion of these definitions.)

The transitive closure of a binary relation R (in the mathematical sense), denoted R^*, is defined by the rules:

$$R \subseteq R^*$$
$$R^* \circ R \subseteq R^*$$

The set $\mathbb{V}$ of variables is the set $\{a_1, a_2, a_3, \ldots\}$ of subscripted "a"s. The set $\mathbb{Y}$ of symbols is $\mathbb{V} \cup \mathbb{N}$. We associate a natural ordering on $\mathbb{Y}$ as follows: $\mathbb{N}$ has its usual ordering; $\mathbb{V}$ is ordered by index value, and every element of $\mathbb{N}$ is less than every element of $\mathbb{V}$. A tableau T of degree m is an N+2-tuple $\langle B,S,T_1,\ldots,T_N\rangle$ such that $S \in \mathbb{Y}^m$, for each $i=1,\ldots,N$, $T_i \subseteq \mathbb{Y}^{\deg(R_i)}$, every variable in S appears in some T_i, and B is a boolean function on the symbols in $T_1,\ldots,T_N$. S is called the summary. We call B the LT-matrix (less-than matrix) because it is intended to represent the "less-than" relations between variables and since it can be considered to be a boolean matrix. We will treat B as a matrix, a boolean function or a binary relation as needed, and will often use the equivalent notation "$B(x,y)=1$" or "xBy" or "$(x,y) \in B$". We consider the empty tableau, $\langle\emptyset,\ldots,\emptyset\rangle$, to be a tableau of any degree.

A tableau set of degree m is a finite set of tableaux of degree m.

If X is a tuple, a tuple set, a tableau or a tableau set, we let $\mathbb{Y}(X)$ denote the set of symbols occurring in X.

A <u>valuation</u> r for tableau T and instance $I = \langle D, O, I_1, \ldots, I_N \rangle$ is a function $\mathbb{Y}(T) \rightarrow D$ which is the identity on $\mathbb{N} \subseteq \mathbb{Y}$. Valuations can be extended to functions on tuples and functions on sets of tuples by component-wise and element-wise extension.

Given a tableau $T = \langle B, S, T_1, \ldots, T_N \rangle$, if B and $<_{\mathbb{N}}$ are union compatible, i.e., if $(B \cup <_{\mathbb{N}})^*$ is asymmetric, then T determines an instance $I = \langle D, O, I_1, \ldots, I_N \rangle$ by defining $D = \mathbb{N} \cup Y(T)$, $O = (B \cup <_{\mathbb{N}})^*$, and $I_i = T_i$, $i=1,\ldots,N$. We will often simply consider T itself to be an instance.

A tableau $T = \langle B, S, T_1, \ldots, T_n \rangle$ may also be considered to be a function $\mathbb{I} \rightarrow \mathbb{N}^{\deg(T)}$ by defining:

$$T(I) = \{r(S) : r \text{ is a valuation for } T, \forall i \; r(T_i) \subseteq I_i, \; r(B) \subseteq O\}$$

It is easy to see that if T can be an instance, then $S \in T(T)$.

A tableau set $Y = \{T_1, \ldots, T_k\}$ may be considered to be a function by defining:

$$Y(I) = T_1(I) \cup \ldots \cup T_k(I).$$

As with expressions, we want to consider two versions of "<u>contained in</u>" for tableaux and for tableau sets. We will write $T_1 \subseteq T_2$ if $T_1(I) \subseteq T_2(I)$ for all instances I. We will write $T_1 \ll T_2$ if $T_1(S) \subseteq T_2(S)$ for all states S. The same notation will also be used for tableau sets.

If $T_1 \subseteq T_2$ and $T_2 \subseteq T_1$, we will write $T_1 \equiv T_2$. If $T_1 \ll T_2$ and $T_2 \ll T_1$, we will write $T_1 \between T_2$.

Next, we want to demonstrate that there is a one-to-one correspondence between elements of **E** and tableau sets.

If e is an expression and T a tableau set, we write $e \equiv T$ if $e(I) = T(I)$ for all instances I. We define a transformation τ from expressions to tableau sets such that $e \equiv \tau(e)$ for all expressions e:

(1) For $R_i \in \mathbb{E}$ of degree m, $\tau(R_i)$ is $\{T\}$, where T is the tableau whose LT-matrix is identically $\emptyset$, whose summary is $\langle a_1,\ldots,a_m\rangle$, whose i-th component is $\{\langle a_1,\ldots,a_m\rangle\}$ and whose other components are empty.

(2) For $\{c\} \in \mathbb{E}$, $\tau(\{c\})$ is $\{T\}$, where T is the tableau whose LT-matrix is identically $\emptyset$, whose summary is $\langle c\rangle$, and whose other components are empty.

(3) For a projection, e[X], if $\langle B,S,T_1,\ldots,T_N\rangle \in \tau(e)$, then $T \in \tau(e[X])$, where T has the form $\langle B,S[X],T_1,\ldots,T_N\rangle$, i.e., the summary is projected on the domains in X, and other components are the same.

(4) For an equi-restriction, e[X=Y], if $\langle B,S,T_1,\ldots,T_N\rangle \in \tau(e)$, where $S = \langle s_1,\ldots,s_m\rangle$, then $T \in \tau(e[X=Y])$, where T is obtained as follows: If s_X and s_Y are the same symbol, then $T = \langle B,S,T_1,\ldots,T_N\rangle$. If s_X and s_Y are unequal constants, then T is the empty tableau. Otherwise assume that the symbol s_X precedes the symbol s_Y in the natural ordering. Then T is obtained by replacing all occurrences of s_Y by s_X. (If we consider B a matrix this means OR-ing the s_Y-row into the s_X-row, OR-ing the s_Y-column into the s_X-column, and removing the s_Y-column and s_Y-row.)

For a less-than-restriction, e[X<Y], if

$\langle B,S,T_1,\ldots,T_N\rangle \in \mathfrak{T}(e)$, where $S = \langle s_1,\ldots,s_m\rangle$, $T \in \mathfrak{T}(e[X<Y])$, where T is obtained as follows: If s_X and s_Y are distinct constants, then T is the empty tableau. Otherwise the LT-matrix for T is obtained from B by setting $B(s_X,s_Y) = 1$. The other components of T are $S,T_1,\ldots,T_N$, unchanged.

(5) For cross product, $e_1 \times e_2$, suppose $T_1 = \langle B_1,S_1,T_{11},\ldots,T_{1N}\rangle \in \mathfrak{T}(e_1)$ and $T_2 = \langle B_2,S_2,T_{21},\ldots,T_{2N}\rangle \in \mathfrak{T}(e_2)$. Let v be the largest variable subscript in T_1 and let m be the smallest variable subscript in T_2. Define the renaming function $g:\mathbb{Y}\to\mathbb{Y}$ by $g(a_j) = a_{j+v+1-m}$, and $g(n) = n$ for $n \in \mathbb{N}$. This maps variables of T_2 to the set of variables with the smallest subscripts which is disjoint from the set of variables in T_1. Let T' be the tableau defined by:

$$B' = B_1 \cup g(B_2)$$
$$S' = S_1\hat{\ }g(S_2)$$
$$T'_i = T_{1i} \cup g(T_{2i})$$

Here '^' denotes concatenation. Then $T' \in \mathfrak{T}(e_1 \times e_2)$.

(6) For union, $\mathfrak{T}(e_1 \cup e_2) = \mathfrak{T}(e_1) \cup \mathfrak{T}(e_2)$.

Lemma 1. For all $e \in \mathbb{E}$, $e \equiv \mathfrak{T}(e)$.

Proof. The proof is by induction on the number of operators in the expression. See [Klug]. □

The existence of the reverse transformation is stated by the next theorem.

Theorem 3. For every tableau set Y there is an expression $e \in \mathbb{E}$ such that $e \equiv Y$.

Proof. A very formal proof would require considerable notational machinery, so we argue informally here.

If $Y = \{T_1,\ldots,T_k\}$, and we get expressions e_i with $e_1 \equiv T_i$, $i=1,\ldots,k$, then we will have $Y \equiv e_1 \cup \ldots \cup e_k$. Hence we need only consider a tableau $T = \langle B,S,T_1,\ldots,T_N \rangle$. We build up e in several steps. First, we build a cross product c which has a term R_i for each row in T_i $(i=1,\ldots,N)$. Then we add an equi-restriction to c for every pair of occurrences (excluding B and S) of the same variable such that if if column A of the m-th row of T_i and column B of the n-th row of T_j are the same variable, then column A of the m-th occurrence of R_i in c is equated to column B of the n-th occurrence of R_j in c. For every occurrence of a constant symbol 'k' in T (excluding B and S) we add an equi-selection term to c such that if 'k' occurs in column A of the m-th row of T_i, then the selection term refers to column A of the m-th occurrence of R_i in c. For every pair of variables x,y in T such that $B(x,y) = 1$ we add a less-than-restriction term to c such that if x occurs in column A of the m-th row of T_i and y occurs in column B of the n-th row of T_j, then the restriction refers to column A of the m-th occurrence of R_i in c and column B of the n-th occurrence of R_j in c (any one such pair of occurrences will do). If one of x,y is a constant symbol, we construct a less-than-selection term similarly. Finally, we add a projection corresponding to the summary of T. If the summary contains a constant 'k', then we add a term {k} to the cross product and add an element to the projection list for this term. For every variable in the summary we add an element to the projection list which is determined by finding any occurrence of this variable in the T_i's and using the corresponding column of c. □

We now have an equivalence between tableau sets and expressions in $\mathbb{E}$. To test for well-formed transactions and for legal histories, we need some computational procedures for tableau sets. The concepts of "chase" and of "containment mapping" will be the appropriate ones.

A <u>chase</u> consists of a sequence of transformations on a tableau set which preserves equivalence. In this paper, we consider transformations determined by schema FDs and ones which manipulate the LT-matrix.

The rules for changing the LT-matrix need to infer all "less-than" relationships among symbols of a tableau. Since it is possible to have, say, $B(a_1,3)=1$ and $B(4,a_2)=1$, but $B(3,4)=\emptyset$, we need to include the order on natural numbers in these rules. Let $\mathbb{N}(T)$ be the constants in tableau $T = \langle B,S,T_1,\ldots,T_N \rangle$, and let $<_{\mathbb{N}(T)}$ be $<_{\mathbb{N}} \sqcap (\mathbb{N}(T) \times \mathbb{N}(T))$. Then we will write B^+ for $(B \cup <_{\mathbb{N}(T)})^*$. This is a closure of B with the ordering on the constants taken into account.

The rules are first defined for tableaux.

<u>F-Rules</u>. For each schema FD $R_i:Z \rightarrow A$ there is an F-rule which is defined as follows: If $T = \langle B,S,T_1,\ldots,T_N \rangle$ and there are t_1, $t_2 \in T_i$ such that $t_1[Z] = t_2[Z]$ and $t_1[A] \neq t_2[A]$, then

(a) If $t_1[A]$ and $t_2[A]$ are unequal constants, replace T by the empty tableau.

(b) Otherwise, if they are unequal symbols s_1, s_2, and s_1 is less than s_2 in the natural orering, replace all the occurrences in T of s_2 by occurrences of s_1 where B is considered to be a set of ordered pairs. (If we consider B a matrix this means OR-ing the s_2-row into the s_1-row, OR-ing the s_2-column into the s_1-column, and removing the s_2-column and s_2-row.)

LT-Rules. If $T = <B,S,T_1,\ldots,T_N>$, replace T by the empty tableau if B^+ has a non-zero diagonal. Otherwise replace T by $<B^+,S,T_1,\ldots,T_N>$.

For a tableau set Y, apply the above rules to the elements of Y.

The transformations derived from these rules have the following properties:

Lemma 2. Let T' be the result of applying an F-rule or an LT-rule to T. Then $T \approx T'$.

Lemma 3. A given set of F-rules can be applied to a tableau only a finite number of times.

Lemma 4. If U and V are tableaux obtained from T by application of F-rules and LT-rules such that no rule can be applied to U or V, then U and V are identical.

These lemmas (Proofs are in [Klug].) mean that the following chase function is well-defined:

chase(T) is the final tableau obtained from T by applying all possible F-rules and LT-rules to T. Chase(Y) is the final tableau set obtained from Y by applying all possible F-rules and LT-rules to members of Y.

Some basic properties of the chase function are the following (proofs in [Klug]):

Theorem 4. Let T' = chase(T). Then, T', as an instance, is a state.

Theorem 5. $T \approx chase(T)$.

It has been shown in [ASU2] and in [ChMe] that the '$\subseteq$' relation for "equality tableaux" and "equality tableau" sets can be determined by certain row-preserving functions on symbols. We next generalize this result to inequality tableaux.

A containment mapping f from tableau T_1 to tableau T_2 is a function $\mathbb{Y}(T_1) \rightarrow \mathbb{Y}(T_2)$ which is one-to-one from the summary of T_1 onto the summary of T_2, which is the identity on constants in T_1, and which has the properties that $f(B_1) \subseteq B_2^+$ and $f(T_{1i}) \subseteq T_{2i}$ for $i=1,\ldots,N$.

Note that there are only a finite number of possible containment mappings from one tableau to another.

Theorem 6. Let T_1, T_2 be tableaux not equivalent to the empty tableau. Then

(1) $T_1 \subseteq T_2$ iff there is a containment mapping $f: T_2 \rightarrow T_1$.

(2) If T_1 is a state (considered as an instance), then $T_1 \ll T_2$ iff there is a containment mapping $f: T_2 \rightarrow T_1$.

Proof. (1) Suppose a containment mapping $f: T_2 \rightarrow T_1$ exists. Let I be an instance and suppose $t \in T_1(I)$. There is a valuation $r: \mathbb{Y}(T_1) \rightarrow D$ with $t = r(S_1)$, $r(T_{1i}) \subseteq I_i$ $(i=1,\ldots,N)$, and $r(B_1) \subseteq O$. The valuation $r \circ f$ for T_2 is such that $(r \circ f)(S_2) = r(S_1) = t$, $(r \circ f)(T_{2i}) \subseteq r(T_{1i}) \subseteq I_i$, and $(r \circ f)(B_2) \subseteq r(B_1^+) \subseteq O$. $(r(B_1) \subseteq O$ implies $r(B_1^+) \subseteq O)$. Thus $t \in T_2(I)$, and $T_1 \subseteq T_2$.

Suppose $T_1 \subseteq T_2$. Then with T_1 as an instance, $S_1 \in T_1(T_1)$, and so $S_1 \in T_2(T_1)$. There is then a valuation r such that $S_1 = r(S_2)$,

$r(T_{2i}) \subseteq T_{1i}$, and $r(B_2) \subseteq (B_1 \cup <_{\mathbb{N}})^*$. The last property can, in fact, be written $r(B_2) \subseteq B_1^+$. Thus r is a containment mapping from T_2 to T_1.

(2) From part (1), we know that existence of a containment mapping $f:T_2 \rightarrow T_1$ implies $T_1 \subseteq T_2$, and $T_1 \ll T_2$ always follows. If T_1 is a state and $T_1 \ll T_2$, Then $S_1 \in T_1(T_1)$ implies $S_1 \in T_2(T_1)$. We proceed as above to get the containment mapping. □

Corollary. For any tableaux T_1, T_2, $T_1 \ll T_2$ iff there is a containment mapping $f:T_2 \rightarrow \text{chase}(T_1)$.

Theorem 7. Let Y_1 and Y_2 be tableau sets.

(1) $Y_1 \subseteq Y_2$ iff there is a containment mapping from each element of Y_2 to some element of Y_1.

(2) $Y_1 \ll Y_2$ iff there is a containment mapping from each element of Y_2 to the chase of some element of Y_1.

Theorem 8. Let Y be a tableau set, and let $Y' = \text{chase}(Y)$. Then $Y \equiv \emptyset$ if and only if Y' consists of the empty tableau.

Proof. If Y' contains only the empty tableau, then $Y \equiv \emptyset$ since $Y(S) = Y'(S) = \emptyset$ for all states. If Y' contains a nonempty tableau T, then Y is not u.e. since $\emptyset \neq T(T) \subseteq Y'(T)$, and T as an instance is a state. □

In defining transactions and locking in the previous section, we needed to derive expressions which pick out the "read set" of an operation. We now give the definition of read set in terms of tableaux and we give the proof of Theorem 1.

Let $T = \langle B,S,T_1,\ldots,T_N\rangle$ be a tableau, and write T_{ij} for the j-th row of T_i (in some arbitrary ordering of the rows). For each $i=1,\ldots,N$ and for each $j=1,\ldots,$#rows of T_i, let RS_{ij} be the tableau $\langle B,T_{ij},T_1,\ldots,T_N\rangle$. Then RS_i is the tableau set $\{RS_{ij} : j=1,\ldots,\#\text{ rows of } T_i\}$. The <u>read set</u> for T, RS(T), is the N-tuple $\langle RS_1,\ldots,RS_N\rangle$. If Y is a tableau set $\{T_1,\ldots,T_n\}$, we define RS(Y) to be the componentwise union of $\{RS(T_1),\ldots,RS(T_n)\}$.

It is not hard to see that the definition of read set for tableaux is equivalent to the one given for expressions.

The two important properties of read sets that justify its use in our locking scheme are that a read set for a tableau set Y can be applied to an instance without changing the value of Y on that instance, and that the read set is the "smallest" function on instances which has this property (if Y itself is optimal). The formal statements are now given.

A tableau T is <u>optimal</u> if every containment mapping $f:T\to T$ is one-to-one and onto. (See [ASU1] and [ChMe] for justification of this definition.) A tableau set Y is optimal if each element of Y is optimal and if $Y \not\equiv Y-\{T\}$ for every T in Y.

<u>Lemma</u> <u>5</u>. Let $T = \langle B,S,T_1,\ldots,T_N\rangle$ be a tableau with read set $\langle RS_1,\ldots,RS_N\rangle$. Then for all instances I and $j=1,\ldots,N$,

$$RS_j(I) = \cup\ \{r(T_j) : r \text{ is a valuation, } r(T_i) \subseteq I_i,\ i=1,\ldots,N,\ r(B) \subseteq O\}$$

Proof. Left to the reader.

<u>Theorem 9</u>. Let T be a tableau with read set $\langle RS_1,\ldots,RS_N\rangle$.

(1) For all instances I, $T(I) = T(RS(I))$.

(2) Let $F = \langle F_1,\ldots,F_N\rangle$ be an N-tuple of tableau sets with the properties that $F_i(I) \subseteq I_i$ $(i=1,\ldots,N)$ for all instances I, and $T(F(I)) = T(I)$ for all instances I. If T is optimal, then $RS_i \subseteq F_i$, $i=1,\ldots,N$.

(3) If T' is a tableau of degree $deg(R_j)$ then for all instances I, if $RS_j(I) \cap T'(I) = \emptyset$, then for $i=1,\ldots,N$, $RS_i(I') = RS_i(I'') = RS_i(I)$, where I', I" are such that $I'_j = I_j \cup T'(I)$, $I''_j = I_j - T'(I)$, and $I'_k = I''_k = I_k$ for $k \neq j$.

Proof. (1) By the previous lemma, it is easy to see that $RS_i(I) \subseteq I_i$, $i=1,\ldots,N$. Hence $T(RS(I)) \subseteq T(I)$. Now suppose $t \in T(I)$. There is a valuation r such that $t = r(S)$, $r(T_i) \subseteq I_i$ and $r(B) \subseteq O$. Then we also have $t \in T(I')$, where $I' = \langle D,O,r(T_1),\ldots,r(T_N)\rangle$. Now $I' \subseteq RS(I)$ by the lemma, so $T(I') \subseteq T(RS(I))$ and $t \in T(RS(I))$.

(2) Each RS_i is a tableau set, and it is sufficient to show that $RS_{ij} \subseteq F_i$ for every member tableau RS_{ij} of RS_i. We will verify this by finding a containment mapping $f:F' \to RS_{ij}$ where is F' is some member tableau of F_i.

Considering T as an instance, we have $F_j(T) \subseteq T_j$ for $j=1,\ldots,N$. Also, since $S \in T(T) = T(F(T))$, we have a valuation r such that $S = r(S)$, $r(T_j) \subseteq F_j(T)$ $(j=1,\ldots,N)$ and $r(B) \subseteq B^+$. Combining, we get $r(T_j) \subseteq T_j$ for $j=1,\ldots,N$, so r is a containment mapping $T \to T$. Since T is optimal, r must be onto: $r(T_j) = T_j$. Then $T_j = r(T_j) \subseteq F_j(T) \subseteq T_j$, so $F_j(T) = T_j$. In particular, $T_{ij} \in F_i(T)$. There is tableau F' in F_i with $T_{ij} \in F'(T)$. As we have seen before, this means there is a containment mapping $F' \to RS_{ij}$ since T_{ij} is the summary of

RS_{ij}.

(3) We have the following formulas from the lemma:

$$\begin{aligned} RS_j(I) &= \cup \{r(T_j) : r(T_i) \subseteq I_i,\ i=1,\ldots,N,\ r(B) \subseteq O\} \\ RS_j(I') &= \cup \{r(T_j) : r(T_i) \subseteq I'_i,\ i=1,\ldots,N,\ r(B) \subseteq O\} \\ RS_j(I'') &= \cup \{r(T_j) : r(T_i) \subseteq I''_i,\ i=1,\ldots,N,\ r(B) \subseteq O\} \end{aligned}$$

Hence, to show that RS_j has the same value on I, I' and I", it sufficient to show that for any valuation r and any $i=1,\ldots,N$, $r(T_i) \subseteq I_i$ iff $r(T_i) \subseteq I'_i$ iff $r(T_i) \subseteq I''_i$. For $i \neq j$, this is clear. If $r(T_j) \subseteq I_j$, then $r(T_j) \sqcap T'(I) = \emptyset$. Therefore, $r(T_j) \subseteq I_j \cup T'(I)$ and $r(T_j) \subseteq I_j - T'(I)$. If $r(T_j) \subseteq I_j - T'(I)$, then, clearly, $r(T_j) \subseteq I_j$. If $r(T_j) \subseteq I_j \cup T'(I)$, we must have $r(T_j) \subseteq I_j$ since $r(T_j) \sqcap T'(I) \subseteq RS_j(I) \sqcap T'(I) = \emptyset$. □

<u>Theorem 10</u>. Let Y a tableau set with read set $\langle RS_1,\ldots,RS_N\rangle$.

(1) Then for all instances I, $Y(I) = Y(RS(I))$.

(2) Let $F = \langle F_1,\ldots,F_N\rangle$ be an N-tuple of tableau sets with the properties that $F_i(I) \subseteq I_i$ $(i=1,\ldots,N)$ for all instances I, and $Y(F(I)) = Y(I)$ for all instances I. If Y is optimal, then for each $i=1,\ldots,N$, $RS_i \subseteq F_i$.

(3) If T' is a tableau of degree $\deg(R_j)$ then for all instances I, if $RS_j(I) \sqcap e'(I) = \emptyset$, then $RS_i(I') = RS_i(I'') = RS_i(I)$, where I', I" are such that $I'_j = I_j \cup T'(I)$, $I''_j = I_j - T'(I)$, and $I'_k = I''_k = I_k$ for $k \neq j$.

Proof. (1) This follows from part (1) of the last theorem.

(2) Let $Y = \{T_1,\ldots,T_k\}$. The condition $Y(F(I)) = Y(I)$ we may write as $T_1(F(I)) \cup \ldots \cup T_k(F(I)) = T_1(I) \cup \ldots \cup T_k(I)$. In particular, this holds for T_i as an instance. Then $S_i \in T_i(T_i) \subseteq T_1(F(T_i)) \cup \ldots$

$\cup\ T_k(F(T_i))$ so there is some j with $S_i \in T_j(F(T_i))$. But with $T_j(F(T_i)) \subseteq T_j(T_i)$ we have $S_i \in T_j(T_i)$. As before, this means $T_i \subseteq T_j$. This contradicts the optimality of Y unless $i = j$. Thus $S_i \in T_i(F(T_i))$. We may proceed as in the previous theorem to get $RS_{T_i j} \subseteq F_j$, $j=1,\ldots,N$. The union gives $RS_j \subseteq F_j$.

(3) The read sets for the tableaux in Y will satisfy the conditions of the last theorem. □

5. Tableau Based Lock Algorithms

In Section 3 we formulated a locking scheme using expressions rather than simple predicates. We allowed the effects of successive locks and unlocks to accumulate with the "lock" and "Lock" functions. We showed that a history is serializable if the transactions are well-formed and two-phased and if the history is legal. The two-phased property can be determined trivially. To test for the well-formed property, we must be able to determine the '$\ll$' relation between an expression to be accessed and a locking expression. To test for the legal property, we must be able to determine the '$\equiv \emptyset$' property for the intersection of an expression to be locked and a lock expression. Without any restrictions on the transactions, the values of the lock function will be arbitrary expressions in the set $\mathbb{E}^-$, and the '$\equiv \emptyset$' and '$\ll$' relations will be undecidable [Solo]. However, when the transactions are two-phased, these relations can be determined (using tableaux). From Section 4 we know how to determine if $e \equiv \emptyset$ or if $e_1 \ll e_2$ when e, e_1 and e_2 are members of $\mathbb{E}$. We now extend these procedures to some simple cases involving the set difference operator. These will be the cases encountered if the transactions are two-phased.

Theorem 11. If TR is two-phased, then lock(TR,M,i,R_j) is equivalent to an expression of the form $e_1 - e_2$, for some $e_1, e_2 \in \mathbb{E}$.

Proof. Before the first unlock of TR, lock(TR,M,i,R_j) is itself a member of $\mathbb{E}$. After the first unlock of TR, there are no more locks, so lock(TR,M,i,R_j) has the form:

$$(\cdots (((e_1 \cup \cdots \cup e_k) - e'_1) - e'_2) - \cdots - e'_m)$$

where e_i and e'_i are in $\mathbb{E}$. This expression is equivalent to the expression:

$$(e_1 \cup \cdots \cup e_k) - (e'_1 \cup \cdots \cup e'_m)$$

□

Theorem 12. Given $e_1, e_2, e_3 \in \mathbb{E}$, $e_1 \ll e_2 - e_3$ iff $e_1 \ll e_2$, and $e_1 \sqcap e_3 \equiv \emptyset$.

Proof. We have the following equivalent statements:

$e_1 \ll (e_2 - e_3)$ is not valid iff

$\exists$ state S, $e_1(S) \underline{\not\subset} (e_2(S) - e_3(S))$ iff

$\exists$ state S, $\exists$ tuple t,

$t \in e_1(S)$ and $t \notin (e_2(S) - e_3(S))$ iff

$\exists$ state S, $\exists$ tuple t,

$t \in e_1(S)$ and $[t \notin e_2(S)$ or $t \in e_3(S)]$ iff

$\exists$ state S, $\exists$ tuple t,

$[t \in e_1(S)$ and $t \notin e_2(S)]$ or

$[t \in e_1(S)$ and $t \in e_3(S)]$ iff

$\exists$ state S,

$e_1(S) \underline{\not\subset} e_2(S)$ or $e_1(S) \sqcap e_3(S) \neq \emptyset$ iff

$e_1 \ll e_2$ is not valid or $e_1 \sqcap e_3$ is not u.e.

□

<u>Theorem 13</u>. Given $e_1, e_2, e_3 \in \mathbb{E}$, $e_1 \sqcap (e_2 - e_3) \equiv \emptyset$ iff $(e_1 \sqcap e_2) \ll e_3$.

Proof. We have the following equivalent statements:

$e_1 \sqcap (e_2 - e_3)$ is not u.e. iff

$\exists$ state S, $\exists$ tuple t, $t \in e_1(S) \sqcap (e_2(S) - e_3(S))$ iff

$\exists$ state S, $\exists$ tuple t,

$t \in e_1(S)$ and $t \in e_2(S)$ and $t \notin e_3(S)$ iff

$\exists$ state S, $\exists$ tuple t,

$t \in (e_1(S) \sqcap e_2(S))$ and $t \notin e_3(S)$ iff

$\exists$ state S, $(e_1(S) \sqcap e_2(S)) \not\subseteq e_3(S)$ iff

$(e_1 \sqcap e_2) \ll e_3$ is not valid.

□

6. Extensions for the Update Operation

In the preceding sections we have developed the idea of expression locks for the operations of insert and delete. We now discuss how updates can be handled.

Updates are operations which change the values of certain domains of selected existing tuples. In our formal model, an <u>update</u> is an operation of the form:

update f R e

Here, R is a schema relation, e is an expression of the same degree as R, and f, the <u>update function</u>, is a unary function from tuples to tuples. For example, "multiply the sal domain by 1.10" is an update function. Intuitively, an such update statement says that all the tuples in R which are selected by e should have the update function f applied to them. Formally, the <u>semantics</u> of the update operation are given by the rule:

$$\begin{aligned}&\text{If } I' = (\text{update } f\ R\ e)(I),\\ &\quad \text{then } I'_j = (I_j - e(I)) \cup f(I_j \cap e(I)),\\ &\quad \text{and } I'_i = I_i \text{ for } i \neq j.\end{aligned}$$

Although it is sometimes said that an update is equivalent to a delete followed by an insert, this is really only true for updates which access tuples by specifying the values of all domains:

$$\begin{aligned}\text{update } f\ R_j\ \{t_1,\ldots,t_n\} \equiv\ & \\ &\text{delete } R_j\ \{t_1,\ldots,t_n\};\\ &\text{insert } R_j\ \{f(t_1),\ldots,f(t_n)\}\end{aligned}$$

(Assuming $\{t_1,\ldots t_n\} \subseteq I_j$.) If this is not the case, the insert operation cannot "find" the tuples anywhere once they are deleted. (We can't do the insert first because could violate FDs.) Thus the update operation must be considered in its own right.

Without any specific information on the nature of the update functions, we cannot extend the algorithms for determining the well-formed property or the legal property. This is because tuples which may be properly locked before the update may not be after it. For example, consider the following transactions:

TR_5: Give every employee whose salary is less than \$15000 a 10% raise.

TR_6: Give every employee whose salary is greater than \$16000 a 5% pay cut.

If TR_5 and TR_6 set the respective locks:

lock X emp emp[sal<15000]

lock X emp emp[sal>16000],

then the locks are disjoint, but the update function for TR_5 can move tuples out of the lock for TR_5 into the lock set of TR_6. (Note, however, that the update function of TR_6 does not move tuples into the lock set of TR_5.) For updates, we therefore to know that the result of the update will still be locked. That is, given an update function f, we need a decision procedure for the relation '$\ll_f$' defined by:

$$e_1 \ll_f e_2 \quad \text{iff} \quad \text{for all states } S,\ f(e_1(S)) \subseteq e_2(S)$$

Then we can extend the definition of well-formed with the rule:

If s_i is "update f R_j e",
then $RS_k \ll \text{lock}(TR,S,i,R_k)$, $k=1,\ldots,N$,
where RS is the read set of e,
$e \ll \text{lock}(TR,X,i,R_j)$ and
$e \ll_f \text{lock}(TR,X,i,R_j)$.

If f is a "constant" update function, i.e., one which can be specified by a rule:

$$A = c,$$

where A is a domain and c is a constant, then $\ll_f$ can easily be determined:

$$e_1 \ll_f e_2 \quad \text{iff} \quad (e_1 \times \{c\})[Z_A] \ll e_2.,$$

Here Z_A is a projection list which includes all domains of e_1 except A, for which it substitutes the domain of {c}. Other update functions, such as the "linear" update functions giving pay raises and pay cuts, would require another approach.

7. An Example

Let us show that the expressions in the introductory example are always disjoint, i.e., that they have an intersection which is u.e. The expression to check is:

```
(employee × dept)
        [edno = dno  &  budget > 1000000]
        [eno,ename,sal,hiredate,edno]
 ⊓
(employee × dept)
        [edno = dno  &  budget < 500000]
        [eno,ename,sal,hiredate,edno]
```

The tableau for this intersection is given in Figure 2. (It can be derived from the definition of the intersection in terms of the basic operations.) By applying the FD rule for the dependency dno→budget, we will replace all occurrences of a_{12} by a_9. Then taking the $^+$-closure of B will give $(a_9, a_9) \in B^+$, so the tableau will be replaced by the empty tableau.

```
B                          S
(1000000,a9)               <a1 a2 a3 a4 a5>
(a12,500000)

|    employee    |         |      dept      |
|----------------|         |----------------|
|a1  a2  a3  a4  a5|       |a5  a7   a8   a9 |
|----------------|         |a5  a10  a11  a12|
                           |----------------|
```

Figure 2. Intersection Tableau

8. Summary and Conclusions

In this paper we have introduced the notion of expression lock. Expression locks are more general than simple predicate locks, and they can allow concurrent execution of transactions which would have to run serially if only simple predicate locks were used. The traditional notions of well-formed transaction and legal history were generalized to handle locks on expressions. Well-formed transactions lock in share mode the "read set" of the expression being read for deletion or insertion. The read set is the smallest possible set of tuples which give the desired set of tuples.

When transactions are two-phased, the relational algebra expressions representing the locks held have a simple form for which the necessary algorithms exist for determining well-formedness (of transactions) and legality (of histories). The algorithms use the tableau technique. To be able to represent a wide class of relational algebra expressions, we extended the notion of tableau by adding a matrix representing "less-than" relationships between variables and between variables and constants.

The algorithms presented offer a practical approach for a database concurrency control method. Although the algorithm for determining well-formed transactions is NP-complete [ASU2], the cost can be amortized over the life of the (canned) transaction. Legality of histories, which must be tested at run time, uses the chase procedure with functional dependency rules, and this can be done in polynomial time [MaMS].

9. Appendix

There are two possible lines in the development of inequality tableaux. The one we have given has the following properties:

(A) (1) The inequality relations in a tableau are given by an arbitrary boolean function on the symbols in the tableau.

(2) In the transformation from expressions to tableau sets, new LT-matrices are obtained by OR-ing rows and columns only.

(3) Containment mappings $f:T_1 \to T_2$ must have the property that $f(B_1) \subseteq B_2^+$.

(4) There is a separate chase rule for manipulation of the LT-matrices which consists of computing the "$^+$-closure".

Another possibility is to require that the LT-matrices be strict orders (asymmetric and transitive). This approach would have the following properties:

(B) (1) The LT-matrix of a tableau must always be an asymmetric transitive order.

(2) Any step in the transformation from expressions to tableau sets which alters the LT-matrix must compute the transitive closure and must replace the tableau by the empty tableau if the closure contains a non-zero diagonal.

(3) Containment mappings $f:T_1 \to T_2$ are homomorphisms in the traditional algebraic sense: The requirement on the LT-matrix is "xBY implies f(x)Bf(y)".

(4) There are no separate rules for LT-matrix manipulation in the chase procedure. Whenever a rule (in our case an F-rule) causes the LT-matrix to change, it must be replaced by the $^+$-closure, and the whole tableau must be replaced by the empty tableau if a non-zero diagonal element appears.

We chose approach (A) because it has the following advantages:

(1) Simplicity of the definition of tableau.

(2) Simplicity of the transformation from tableaux to expressions.

(3) Separate LT-rules. The $^+$-closure needs to be taken only once at the end of the chase.

We note, however, that there are some disadvantages:

(1) The definition of containment mapping does not correspond to the traditional notion of homomorphism.

(2) Nonempty tableaux may represent the empty function (if the $^+$-closure has a non-zero diagonal), and we cannot write "$S \in T(T)$" without qualifications.

References

[Acke] Ackermann W. "Solvable Cases of the Decision Problem", North Holland, 1962

[ASU1] Aho A.V., Sagiv Y. and Ullman J.D. "Efficient Optimization of a Class of Relational Expressions", ACM Trans. on Database Systems, 4, 435-454 (1979)

[ASU2] Aho A.V. Sagiv Y. and Ullman J.D. "Equivalence of Relational Expressions" SIAM J. Comptng. 8, 2, 218-246 (May 1979)

[BeGo1] Bernstein P.A. and Goodman N. "The Theory of Semi-Joins", Tech. Rep. CCA-79-27, 1979, Computer Corp. of America

[BeGo2] Bernstein P.A. and Goodman N. "Fundamental Algorithms for Concurrency Control in Distributed Database Systems", Tech. Rep. CCA-80-05, Computer Corp. of America, 1980

[ChMe] Chandra A.K. and Merlin P.M. "Optimal Implementation of Conjunctive Queries in Relational Databases", Proc. 9-th Annual Symp. on Theory of Computing, May, 1976, 77-90

[EGLT] Eswaran K.P., Gray J.N., Lorie R.A. and Traiger I.L. "The Notions of Consistency and Predicate Locks in a Database System" CACM 19, 11, pp.624-633

[KlPr] Klug A. and Price R. "Generalized Tableaux for Chasing Expression Constraints" to appear

[Klug] Klug A. "On Inequality Tableaux", to appear

[MaMS] Maier D., Mendelzon A.O. and Sagiv Y. "Testing Implications of Data Dependencies" ACM-TODS 4, 4, 455-469 (1979)

[Papa] Papadimitriou C.H. "The Serializability of Concurrent Database Updates" JACM 26, 631-653 (1979)

[SaYa] Sagiv Y. and Yannakakis M. "Equivalence Among Relational Expressions with the Union and Difference Operator" Fourth International Conference on Very Large Data Bases, West-Berlin, 1978

[Sibl] Sibley E.H.(ed) "Data-Base Management Systems" ACM Computing Surveys, vol. 8, #1

[Solo] Solomon M. "Undecidability of the Equivalence Problem for Relational Expressions" Bell Laboratories, to appear

[Ullm] Ullman J.D. "Principles of Database Systems", Computer Science Press 1980

[WoEd] Wong K.C. and Edelberg M. "Interval Hierarchies and Their Application to Predicate Files" ACM Transactions on Database Systems, 2, 223-232 (1977)