SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

CONCURRENCY CONTROL BY LOCKING*

CHRISTOS H. PAPADIMITRIOU†

Abstract. We present a geometric method for studying concurrency control by locking. When there are only two transactions, our method yields an exact characterization of safe locking policies and also of deadlock-free locking policies. Our results can be extended to more than two transactions, but in that case the problem becomes NP-complete.

Key words. database, concurrency control, transaction, locking, two-phase locking, geometry of locking, serializability, safety, NP-complete problems

1. Introduction. A database consists of a set of named data objects called *entities*. The values of these entities must at any time be related in some ways, prescribed by the *integrity constraints* of the database. When a user accesses or updates a database, she/he may have to violate temporarily these integrity constraints, in order to restore them at some later time, with the specific data changed. For example, in a banking system, there may be no way to transfer funds from an account to another in a single atomic step, without temporarily violating the integrity constraint stating that, say, the sum of all balances equals the total liability of the bank. For this reason, several steps of the interaction of the same user with the database are grouped into a *transaction*. Transactions are assumed to be *correct*, that is, they are guaranteed to preserve consistency when run in isolation from other transactions.

When many transactions access and update the same database concurrently, the consistency of the database may fail to be restored after all transactions have been completed. If, for example, transaction 1 consists of the two steps

$$x := x+1, \qquad x := x-1$$

and transaction 2 of the single step

$$x := 2*x,$$

and the consistency requirement is simply "$x=0$", then executing transaction 2 between the two steps of transaction 1 turns a consistent database into an inconsistent one. This is despite the fact that both transactions are *individually* correct; that is, each preserves database consistency when run alone. We must therefore find ways to prevent such undesirable interleaving without excessively harming the parallelism and overall efficiency of the system. This is the *database concurrency control* problem, already discussed extensively in the literature (see [Pa2], [EGLT1], [EGLT2], [SLR], [BGRP], [Pa1], [KP], [Ya], [SK]).

Since we assume that each transaction is by itself correct, a reasonable goal for the database concurrency control mechanism would be to rearrange the steps of the transactions so that the resulting sequence of steps is *serializable*. This means that its effect is as though the transactions executed without any interleaving, one after the other in some order. Serializability has been widely recognized as the right notion of correctness (e.g. [EGLT1], [SLR], [Pa1]). In fact, in [KP] we show that it is the most liberal notion of correctness possible, when only syntactic information (i.e., the names of the entities accessed at each step) is available. We denote the set of all serializable schedules by *SR*.

* Received by the editors December 15, 1981, and in revised form June 7, 1982.

† Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, and National Technical University of Athens, Greece.

A very common way for implementing concurrency control is *locking*. In this method each entity is equipped with a binary semaphore—its *lock*—and transactions synchronize their operation by locking and unlocking the entities that they access. The lock-unlock steps are inserted in a transaction according to some *locking policy*. A locking policy may have the property that, if all transactions are locked according to it, then any interaction is guaranteed to be serializable. Such a locking policy is called *safe*. A classical example of safe locking policies is the *two-phase locking* (2PL) policy proposed in [EGLT1]. In 2PL a transaction must lock an entity before accessing it, and must not unlock it until after the last access of the entity, *and after the last lock of the transaction is granted*. Thus a transaction has two phases: the *locking phase*, during which the transaction requests (but does not release) locks, and the *unlocking phase*, during which locks are released but not requested. 2PL is a safe locking policy [EGLT1]. Another safe locking policy is the *tree policy* of [SK], in which the entities are arranged in a tree, typically one reflecting some logical or physical structure, and transactions access whole subtrees. An entity may be locked only if its father entity is currently locked. This tree policy can be further generalized to the *digraph policy* and the *hypergraph policy* [Ya]; the latter is shown in [Ya] to be the most general safe locking policy possible, in the sense that *all* safe policies can be considered as special cases of the hypergraph policy.

In [Pa1], [KP] we proposed a measure whereby the performance of concurrency control mechanisms in general—and locking policies in particular—can be evaluated in a uniform setting. This measure is expressed in terms of the class of all sequences of transaction steps that can be the response of the concurrency controller to a stream of execution requests. The richer this class, the fewer unnecessary delays and rearrangements of steps will occur, and the greater the *parallelism* supported by the system.

In this paper we study safe locking policies in a theoretical, unifying way. In § 2 we describe our model and define our terminology.

In §§ 3 and 4 we characterize safe locking policies. We first give a characterization of safety for the case of two transactions. To do so, we employ a geometric methodology reminiscent of that used by Dijkstra for studying *deadlocks* [CES], [CD]. Here we use it in a very different way to study *incorrect completions*. (We ignore deadlocks, as seems reasonable to do in view of their limited significance in this context; see [GLPT].)

Besides its independent interest and elegance, the two-transaction solution is the building block for resolving the general case (§ 4). It turns out that a locking policy defined on $d > 2$ transactions is safe if and only if all of its two-transaction subsystems are safe, plus a combinatorial condition. This combinatorial condition turns out to be NP-complete, but it is simple enough to have some interesting corollaries. For example, all specific locking policies mentioned above can be shown to be safe as immediate consequences of the condition.

Versions of several of the results in this paper—in particular those in § 4—were independently obtained by Mihalis Yannakakis—see the first part of [Ya]. Both our results and those in [Ya] were announced in a joint extended abstract [YPK].

2. Definitions.

A *transaction system* $\tau = \{T_1, \cdots, T_d\}$ is a set of *transactions*. A transaction $T_i = (T_{i1}, \cdots, T_{im_i})$ is a sequence of *actions* or *transaction steps*. Each action T_{ij} has associated with it an *entity*, $x_{ij} \in E$, where E is a set of entities. The x_{ij}'s need not be distinct.

Each action T_{ij} is thought of as the indivisible execution of the following:

$$T_{ij}\colon t_{ij} := x_{ij}$$

$$x_{ij} := f_{ij}(t_{i1}, \cdots, t_{ij}).$$

The first instruction stores the current value of x_{ij} to a local variable t_{ij}, not in E, and the second changes x_{ij} in the most general possible way based on all available local (to the transaction) information. The t_{ij}'s are all distinct. A *schedule* (or *history*) s of τ is a permutation of all steps of τ such that $j<k\leqq m_i$ implies $s(T_{ij})<s(T_{ik})$. The set of all schedules is denoted by S. s is called *serial* if, for all i and $j<m_i$, $s(T_{ij})+1=s(T_{i,j+1})$. Two schedules are *equivalent* if they are equivalent as parallel program schemata with uninterpreted f_{ij}'s; s is *serializable* (notation: $s\in SR$) if it is equivalent to some serial schedule.

Deciding serializability of a schedule s is known to be an NP-complete problem if we distinguish between reading and writing steps [Pa], [PBR], but can be easily done in our model of actions as follows [EGLT1]: Construct a digraph $D(s)$ by associating a node with each transaction T_i and drawing an arc (T_i, T_j) whenever, in the schedule s, T_i updates an entity before T_j does. Then s is serializable if and only if $D(s)$ is acyclic.

A *locked transaction system* $L(\tau)$ is a special augmented version of the (ordinary) transaction system τ. The operator L performing this augmentation is called a *locking policy*. A locking policy transforms each transaction of τ by inserting *lock* x and *unlock* x steps, $x\in E$, according to the following rules:

(1) For each entity x there is at most one *lock* x step in each transaction. If it exists, then there is also a unique subsequent *unlock* x step.

(2) Every access to an entity x is surrounded by a *lock-x–unlock-x* pair.

Let s be a schedule of $L(\tau)$. s is said to be *legal*—notation: $s\in G(L(\tau))$—if and only if between any two occurrences of *lock* x in s there is an occurrence of *unlock* x. Let L^{-1} be the operator that removes all lock-unlock steps; the set $L^{-1}(G(L(\tau)))$—$O(L)$ for short—is the *output set* of the locking policy L, and it captures the essential amount of parallelism supported by L.

We say that a locked transaction system $L(\tau)$ is *safe* if and only if any schedule in $O(L)$ is serializable. We say that it is *deadlock-free* if and only if for any legal *prefix* p of a schedule of $L(\tau)$, there is a *suffix* s such that $p\cdot s\in G(L(\tau))$.

Given a transaction system τ, there are certain well-known locking policies that can be applied to it. One is the *two-phase locking* (2PL) policy [EGLT1]. In it we insert locks surrounding the accesses of all entities in each transaction, subject to the following rule: The last entity to be locked is locked before the first entity is unlocked. Thus, a transaction is divided into two phases: the *locking phase*, during which locks are acquired but not released, and the *unlocking phase*, in which locks are released but not requested. Notice that this does not uniquely define a locking policy; it is in fact a *family* of locking policies. In an extremely conservative interpretation, we could lock all entities before the first step, and unlock them after the last. More reasonably, we could request for entities at the first step that they are accessed, and release locks at the end of the transaction. In fact, it is shown in [KP] that the latter interpretation of 2PL is the best possible concurrency control, when syntactic information is acquired in an incremental, dynamic manner (best in terms of the parallelism allowed). It was first shown in [EGLT1] that 2PL is safe (though not deadlock-free).

If the entities are *unstructured* (that is, transactions access them in all possible permutations of orders and patterns) then 2PL is the best possible locking policy. Suppose, however, that the entities form a tree, and are accessed by transactions as follows:

(a) A transaction accesses a subtree, whose root is the first entity to be accessed (after, of course, it is locked).

(b) After this, when an entity is locked, its parent must be locked and not yet unlocked.

Then this family of locking policies, called the *tree policy*, is shown in [SK] to be both safe and deadlock-free. This holds for the more general *digraph policy* of [Ya]. In fact, the latter is generalized in [Ya] to the *hypergraph policy* which, it is proved, is the most general possible safe policy.

In this paper we are only discussing what is known as the *exclusive* version of locking. There are, however, variants of locking in which locks of different kinds are defined (e.g., shared locks or read-locks, intention locks, etc.). Certain kinds of locks may coexist with others, whereas certain other kinds cannot. In [Pa3] we show that in some respects these more general kinds of locking behave in a way very similar to ordinary exclusive locks. It appears that the results of this paper can be quite easily generalized to these kinds of locks.

3. The geometry of locking.

3. The geometry of locking. Consider a transaction system τ consisting of two transactions, T_1 and T_2. In the coordinate plane (Fig. 1) take the two axes to correspond to T_1 and T_2, and the integer points 1,2, etc., on these axes to correspond to the steps T_{11}, T_{12}, etc. (respectively T_{21}, T_{22}, etc.) of the transactions. A point p may present

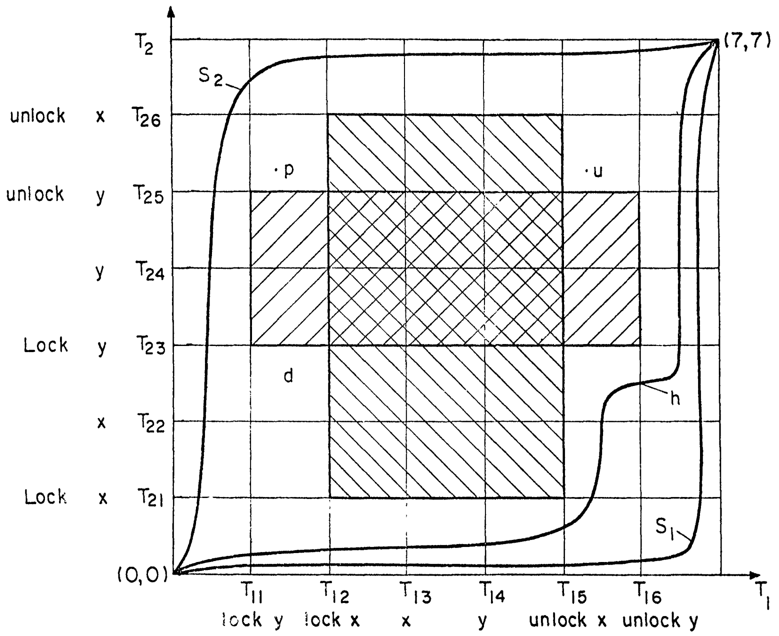


FIG. 1

a possible state of progress made toward the completion of T_1 and T_2. T_1 and T_2 will in general be *locked* transactions and will therefore contain properly nested lock-unlock steps. Each entity x such that both T_1 and T_2 contain a *lock-x–unlock-x* pair, has the effect of creating a *forbidden region* (a rectangle delimited by the grid lines corresponding to the four *lock x* or *unlock x* steps), the points of which represent unreachable states (see Fig. 1). Adding such rectangles to the plane has some consequences. For example, the point u is now unreachable, yet not in any rectangle; in contrast, point d is a state of *deadlock*.

What is the geometric image of a schedule? A schedule is a nondecreasing curve from the point (0, 0) to the point (m_1+1, m_2+1), not passing through any other grid point, nor through any rectangle (e.g., h in Fig. 1). (In fact, a schedule is more precisely a *class* of such curves, all of which cross the same line segments of the grid.) To read the schedule off any such curve, we simply enumerate the grid lines that it intersects. For example, in Fig. 1 $h = T_{11}T_{12}T_{13}T_{14}T_{15}T_{21}T_{22}T_{16}T_{23}T_{24}T_{25}T_{26}$. The two serial schedules are represented by the curves s_1 and s_2 in Fig. 1.

Let h and h' be schedules such that $h = h_1S_1S_2h_2$, $h' = h_1S_2S_1h_2$, and S_1, S_2 are steps *not* updating the same variable. We then write $h \sim h'$. Let $\overset{*}{\sim}$ be the transitive-reflexive closure of $\sim$. The following fact has appeared many times in the literature [EGLT1], [PBR], [Pa1].

LEMMA 1. *Let h, h' be schedules. Then $h \equiv h'$ if and only if $h \overset{*}{\sim} h'$.*

Consider the point set $S = [0, m_1+1] \times [0, m_2+1] - R$, where R is the set of all forbidden rectangles. In other words, S is the relevant nonforbidden region of the plane. Let $\mathscr{C}$ be the set of all nondecreasing curves from $(0, 0)$ to (m_1+1, m_2+1) in S. We can partition these curves into *homotopy classes*. Two curves in $\mathscr{C}$ are *homotopic* if their union can be shrunk into a single point. In Figure 2, for example, h_1 and h_2 are homotopic, whereas h_1 and h_3 are not (since they "enclose" a forbidden rectangle). Homotopy is an equivalence relation in $\mathscr{C}$. Also, if two curves represent the same schedule, then they are homotopic.

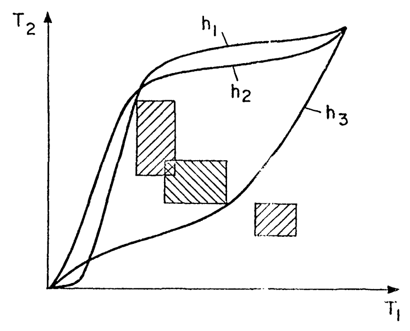


FIG. 2

LEMMA 2. *Two schedules are equivalent if and only if any two corresponding curves are homotopic.*

Proof. We first notice that condition (a) in our definition of locking has as a consequence that every forbidden rectangle contains a grid point (T_{1i}, T_{2j}) such that T_{1i} and T_{2j} update the same entity x; and conversely, any such point is surrounded by a forbidden rectangle. It follows that two curves in $\mathscr{C}$ are homotopic in $[0, m_1+1] \times [0, m_2+1] - R$ if and only if they are homotopic in $[0, m_1+1] \times [0, m_2+1] - P$, where P is the set of all points (T_{1i}, T_{2j}) such that T_{1i} and T_{2j} update the same entity. Now any two curves $h, h' \in \mathscr{C}$ are homotopic in $[0, m_1+1] \times [0, m_2+1] - P$ if and only if h can be transformed into h' via a continuous transformation, avoiding all points in P. Such a continuous transformation can be broken down into finitely many transformations of one of the following two types:

(a) transformations that do not change the schedule represented by the curve (Fig. 3a),

(b) transformations in which the curve crosses a grid point *not in P*. (Fig. 3b.)

Type (a) leaves the corresponding schedule unchanged. Also, type (b) changes the schedule h into a h' such that $h \sim h'$. Therefore, we have from the above discussion that two curves are homotopic if and only if the corresponding schedules satisfy $h'^{*} \sim h$; or, by Lemma 1, if and only if $h \equiv h'$. □

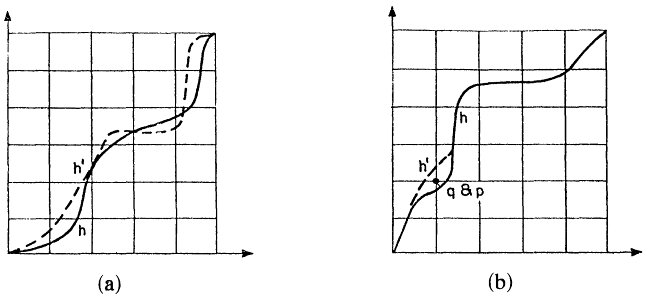


FIG. 3

Since the two serial schedules are s_1 and s_2 of Fig. 1, we conclude from Lemma 2 that nonserializable schedules are exactly those that are homotopic to neither.

THEOREM 1. *A schedule is not serializable if and only if the corresponding curve separates two rectangles.*

Thus h_1 and h_2 in Fig. 2 are serializable schedules, whereas h_3 is not. Hence τ is unsafe.

Certain further geometric concepts help illuminate the concept of *safety* of two-transaction systems. Let R be any subset of the plane, possibly disconnected. We call two points (x_1, y_1) and (x_2, y_2) in the plane *incomparable* if $(x_1-x_2)\cdot(y_1-y_2)<0$ (points p and q in Fig. 4). Then R is said to be *closed* if, for any two incomparable points (x_1, y_1) and (x_2, y_2) that are connected in R, the points (x_1, y_2) and (x_2, y_1) are also in R. The *closure* of R is the smallest closed region that contains R. Notice that closure (R) is always well-defined (see Fig. 4).

LEMMA 3. *If a connected rectilinear*[1] *region R is closed, then R is the region contained between two increasing curves that intersect only at their endpoints.*

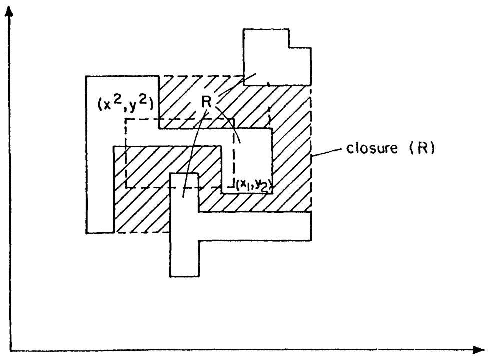
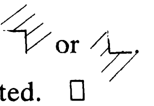


FIG. 4

Proof. The boundary of a rectilinear region R is *not* as in the lemma only if it contains a fragment like



In both cases closedness is contradicted. □

We make here the following helpful observation: The forbidden rectangles corresponding to two locked transactions have the property that each of their edges has a unique ordinate or abscissa. As a consequence, the same is true for the rectangles

[1] A region is called *rectilinear* if it is the union of rectangles with edges parallel to the axes.

comprising the closure of the forbidden region, as the closure of a rectilinear region has rectangles with ordinates and abscissas among those of the original rectangles. A helpful corollary is that components of the closure may not accidentally "touch" at a line, or just a point. If they are disconnected, they are clearly divided by corridors of width one. Therefore, a nondecreasing curve avoiding all components can always be turned to one that avoids all grid points (except (0, 0) and $(m_1+1,\ m_2+1)$).

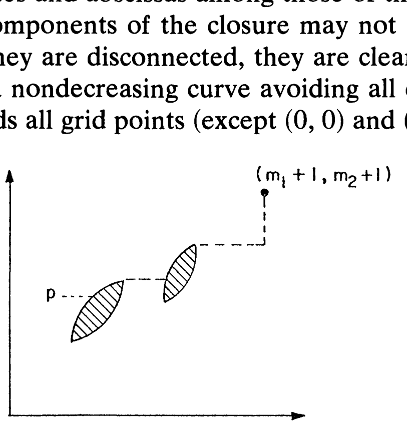


FIG. 5

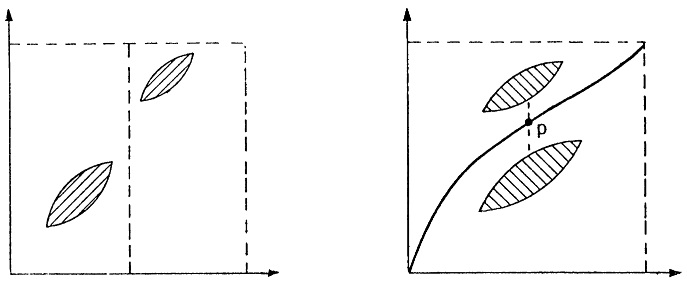
LEMMA 4. *Let R be a closed (possibly disconnected) rectilinear subset of $[0, m_1+1]\times[0, m_2+1]$, and let p be a point in $[0, m_1+1]\times[0, m_2+1]-R$. Then there is a nondecreasing curve from $(0, 0)$ to p and from p to (m_1+1, m_2+1) in $[0, m_1+1]\times[0, m_2+1]-R$.*

Proof. We shall only prove the second claim, as the first is completely symmetric. From p we first go parallel to the x-axis to increasing x's, see Fig. 5. If we do not "hit" a component of R, we are done; otherwise, we follow the nondecreasing curve in the boundary of the component (Lemma 3) and when it ends we again proceed parallel to the x-axis. It is clear that we shall eventually reach the $x = m_1+1$ line. □

THEOREM 2. *τ is safe if and only if the closure of the union of the forbidden rectangles is connected.*

Proof. Suppose that τ is unsafe. Then, by Theorem 1, there is an increasing curve h from $(0, 0)$ to $(m_1+1,\ m_2+1)$ which separates two rectangles r_1 and r_2. Consider the region that consists of all grid squares not touched by h. This region is closed, contains all rectangles and is disconnected, and each of its components contains a rectangle. It follows that the closure of the forbidden region is the union of two nonempty subsets of the two components, and thus it is disconnected.

For the "if" direction, suppose that the closure is disconnected. There are two cases. Either there is a vertical line which hits two components of the closure, or there is not. In the second case (Fig. 6a) there is a vertical line which *separates* two components—and thus a nondecreasing curve, consisting of two horizontal and one vertical



(a) (b)

FIG. 6

line segments, which also separates two components. In the first case, (Fig. 6b) consider this line, and a point p on it that is in between the two components. By Lemma 4, there is a nondecreasing curve from $(0, 0)$ to p and one from p to (m_1+1, m_2+1). Their union is a nondecreasing curve that separates the two components. □

In a rectilinear region R we can define the *lower-left boundary*, LLB (R), to be the union of all horizontal segments that are lower boundaries of R, together with all vertical segments that are left boundaries of R (that is, LLB (R) is the set of all points (x, y) such that $(x+\delta, y+\varepsilon)\in R$ whereas $(x-\delta, y-\varepsilon)\notin R$, for all $\delta, \varepsilon>0$). The proof of the following result is now straight forward.

THEOREM 3. *τ is deadlock-free if and only if* LLB $(R)\supseteq$ LLB *(closure* (R)*), where R is the union of the forbidden rectangles.*

Theorems 2 and 3 lead to fast algorithms for the solution of the safety and deadlock problems for locked transaction systems [LP]. There are other insights that are offered by this geometric viewpoint. For example, the correctness of 2PL has now become very intuitive. 2PL says that all rectangles must contain the point p whose projections p_1 and p_2 mark the phase-shift points of the two transactions (see Fig. 7). Thus the rectangles are certainly connected, so is their closure, and the correctness of 2PL follows immediately from Theorem 2.

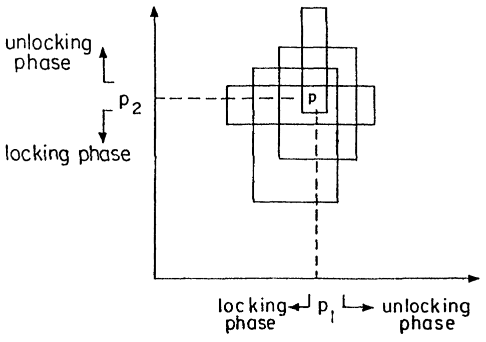


FIG. 7

4. More than two transactions. Consider now a set of $d>2$ locked transactions $\tau=\{T_1, \cdots, T_d\}$. We wish to study the problem of safety of τ. If any subset of τ with two transactions is unsafe, then clearly so is τ. Define the graph $G(\tau)=(\tau, A)$ with transactions as nodes, and with $[T_i, T_j]\in A$ if and only if T_i and T_j update a common entity. From our assumption above it follows that, for any $[T_i, T_j]\in A$, the closure of the forbidden region on the (T_i, T_j)-plane is connected. Therefore, any schedule h of τ will have a projection on the (T_i, T_j)-plane that is either equivalent to T_iT_j or equivalent to T_jT_i (see Fig. 8). In the first case we write $T_i <_h T_j$, and in the latter $T_j <_h T_i$.

LEMMA 5. *h is serializable if and only if $(\tau, <_h)$ is acyclic.*

Proof. The lemma follows by observing that $(\tau, <_h)$ is exactly the digraph constructed for testing the serializability of h in § 2. □.

Therefore, for τ to be safe, for each directed cycle that corresponds to an undirected cycle in $G(\tau)$ there must be a reason why no h exists that has this same cycle in the graph of $<_h$. Intuitively, the reason is that there is a contradiction in the order in which the curve of h intersects the prisms with bases marked $P_1, P_2, \cdots, Q_1, Q_2$ in Fig. 8. This is captured as follows: With each pair $([T_i, T_j], [T_j, T_k])$ of edges in A we associated a digraph B_{ijk}. The vertices of this

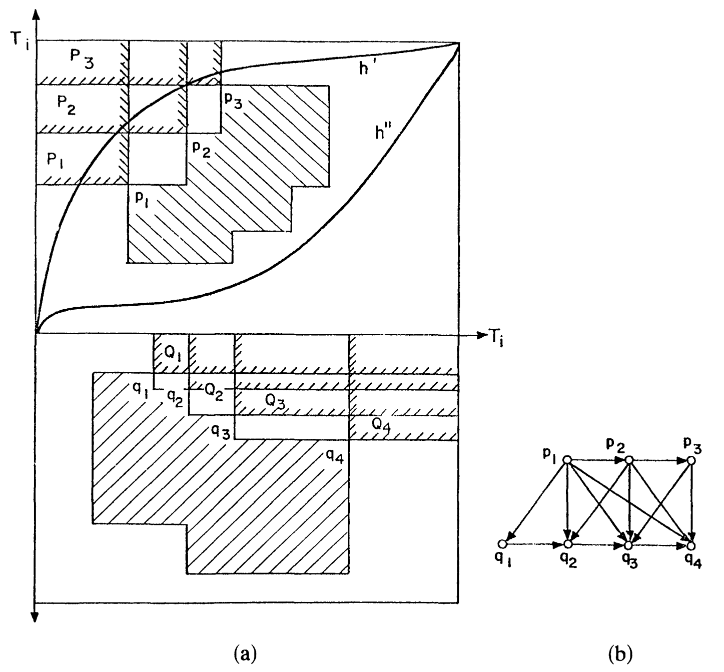


FIG. 8

digraph are the vertices p_m, q_n of the P_m and Q_n regions (see Fig. 8). There is an arc from u to v if and only if (a) either u is a p_m or v is a q_n (or both), and (b) the T_j-coordinate of u is smaller than the T_j-coordinate of v. The construction is illustrated in Fig. 8b. Finally, if C is a directed cycle corresponding to a simple undirected cycle in $G(\tau)$, we let B_C be the union of all B_{ijk} digraphs for all consecutive triples (T_i, T_j, T_k) of C. The result is the following:

THEOREM 4. *τ is safe if and only if*

(a) *the restriction of τ to any two transactions is safe, and*

(b) *for all directed cycles C corresponding to undirected simple cycles of $G(\tau)$, the digraph B_C has a cycle.*

Proof. We shall assume throughout the proof that condition (a) holds, as otherwise both directions are easy. Suppose that τ is unsafe, and yet (b) holds. Then there is a schedule h such that $<_h$ has a simple cycle C. By condition (b), B_C has a cycle. Consider, however, the order whereby h enters (equivalently, leaves) the different prisms P_i, Q_j, etc., appearing in B_C. This order must be, by the construction of B_C, consistent with the arcs in B_C. This, however, is a contradiction, since B_C was assumed to have a cycle, and therefore to be inconsistent with any linear order.

Conversely, suppose that (b) does not hold; suppose, that is, that there is a directed cycle C such that B_C is acyclic. Let $(p_1, q_1, \cdots)$ be a linear order consistent with B_C. We construct a nonserializable schedule h of τ as follows: We first arrange all transactions not appearing in C in some serial order. We then execute the transactions in C by starting with all program counters to 0, and by successively considering the next point in the order. If this point lies on the (T_i, T_j)-plane, where $(T_i, T_j) \in C$, then we proceed by arranging the steps of T_j and then the steps of T_i that bring the state to the point considered. Since the order is consistent with B_C, this will always be possible; after the last point we somehow schedule all remaining steps. It is easy to see that, if h is the resulting schedule, then $(\tau, <_h)$ contains the cycle C, and therefore h is not serializable. □

Based on Theorem 4, we obtain extremely easy proofs of correctness of several locking policies:

COROLLARY 1. *Any transaction system obeying* 2PL *is safe.*

Proof. That the two-transaction subsystems of any transaction system that obeys 2PL is safe follows trivially from Theorem 2, as discussed in the end of the previous section. Property (b) of Theorem 4 follows from the fact in 2PL the graphs B_{ijk} are complete bipartite. □

COROLLARY 2. *Any transaction system obeying TP is safe.*

Proof. Since the common variables of any two transactions form a rooted tree in *TP*, condition (a) of Theorem 4 is trivial. Condition (b) also follows easily from the tree structure. □

Consider now the special case in which any two transactions have at most one variable in common—or, equivalently, the closure of the forbidden region on all planes is either empty or rectangular.

COROLLARY 3. *Under the above assumption τ is safe if and only if each of the restrictions of τ to every biconnected component of $G(\tau)$ obeys* 2PL.

Another consequence of Theorem 4 is an algorithm for checking a transaction system for safety.

COROLLARY 4. *Checking a transaction system τ for safety can be done in time polynomial in the number of minimal cycles of $G(\tau)$.*

In general, of course, $G(\tau)$ will have an exponential number of minimal cycles, and thus Corollary 7 does not imply a genuine polynomial-time algorithm. In fact, such an algorithm is quite unlikely in view of the next result:

THEOREM 5. *Testing a transaction system for nonsafety is* NP-*complete.*

Proof. Recently Fortune, Hopcroft and Wyllie [FHW] showed that almost all digraph homeomorphism problems—specifically, those whose pattern is not a rooted tree of depth one—are NP-complete. From their results one immediately deduces that the following problem is NP-complete:

"Given a digraph $D=(V, A)$ and four arcs (v_1, v_2), (v_2, v_3), (u_1, u_2), $(u_2, u_3) \in A$, is there a simple cycle C of D such that $C=(v_1, v_2, v_3, \cdots, u_1, u_2, u_3, \cdots, v_1)$?"

We shall reduce this problem to the nonsafety problem. Given a digraph $D=(V, A)$ and four arcs (v_1, v_2), (v_2, v_3), (u_1, u_2), (u_2, u_3) we shall construct a locked transaction system τ such that C exists in D if and only if τ is unsafe. τ has one transaction T_v for each node v of D. Let us assume that for no nodes $u, v \in V$, does A contain both (u, v) and (v, u)—that is, $A \cap A^{-1} = \emptyset$ (it is easily seen that this is not a loss of generality, since a new node can always be placed in the middle of an arc). We shall next describe the graphs B_{uvw} for (u, v), $(v, w) \in A$. Unless $(u, v, w) = (u_1, u_2, u_3)$ or (v_1, v_2, v_3), B_{uvw} is the graph shown in Fig. 9a, realized as shown in Fig. 9b. For these two triples, B_{uvw} is as shown in Fig. 9c, realized as in Fig. 9d.

Consider now any simple cycle in D. The corresponding union of the B_{uvw}'s will consist of concatenations of graphs such as in Figs. 9a and 9c. It is easy to see that such a concatenation is acyclic only if it contains at least two copies of Fig. 9c, that is, only if C passes through (v_1, v_2, v_3) and (u_1, u_2, u_3). Conversely, consider any directed cycle C, whose undirected version is a simple cycle in $G(\tau)$, such that B_C is acyclic. If, however, any arc (u, v) of C is not in A—that is, $(v, u) \in A$—then B_C will contain two complete bipartite graphs corresponding to (u, v), its predecessors and its successor edge in C (by the construction in Fig. 9b, d). It follows easily that B_C could not be acyclic. Thus all arcs in C are also in A, and in fact, since B_C was acyclic, all four (v_1, v_2), (v_2, v_3), (u_1, u_2), and (u_2, u_3) are in C. The theorem follows. □

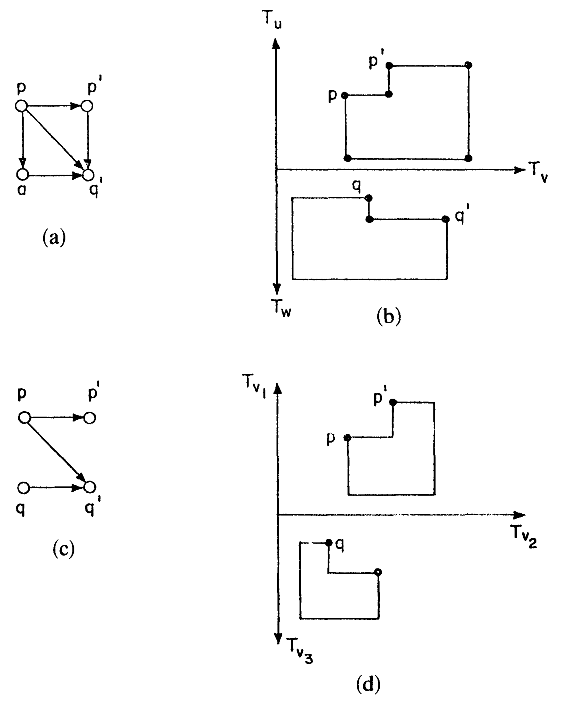


FIG. 9

Consequently, we can view Corollary 3 as suggesting an efficient algorithm for a special case of the NP-complete problem of nonsafety—namely, the case in which any two transactions interact by at most one entity. The result is tight, since the proof of Theorem 5 (see Figs. 9b, d) suggests that the problem remains NP-complete when two entities per pair of transactions are allowed.

REFERENCES

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

[BGRP] P. A. BERNSTEIN, M. GOODMAN, J. B. ROTHNIE AND C. H. PAPADIMITRIOU, *Analysis of Serializability of SSD-1: A system of distributed databases (The fully redundant case)*, IEEE Trans. Software Engng, SE-4 (1978), pp. 154–168.

[CD] E. G. COFFMAN, JR. AND P. J. DENNING, *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[CES] E. G. COFFMAN, JR., M. J. ELPHICK AND A. SHOSHANI, *Systems deadlock*, Comput. Surveys, 3 (1971), pp. 67–68.

[EGLT1] K. P. ESWARAN, J. N. GRAY, R. A. LORIE AND I. L. TRAIGER, *The notions of consistency and predicate locks in a database system*, Comm. ACM, 19 (1976), pp. 624–633.

[EGLT2] ———, *On the notions of consistency and predicate locks*, IBM Research Report RJ 1487, 1974.

[FHW] S. FORTUNE, J. E. HOPCROFT AND J. WYLLIE, *The directed subgraph homeomorphism problem*, Theoret. Comput. Sci. 10 (1980), pp. 111–121.

[GJ] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Fransisco, 1978.

[GLPT] J. N. GRAY, R. A. LORIE, G. R. PUTZOLU AND I. L. TRAIGER, *Granularity of locks and degrees of consistency in a shared data base*, IBM Research Report RJ 1654, 1975.

[Ka] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum, New York, 1972, pp. 85–103.

[KM] R. M. KARP AND R. E. MILLER, *Properties of a model for parallel computations: Determinacy, termination and queuing*, SIAM J. Appl. Math., 14 (1966), pp. 1390–1410.

[KP] H. T. Kung and C. H. Papadimitriou, *An optimality theory of concurrency control for databases*, Proc. ACM-SIGMOD Conference, 1979,

[LW] Y. E. Lien and P. H. Weinberger *Consistency, concurrency, and crash recovery*, Proc. ACM-SIGMOD Conference, 1978,

[LP] W. Lipski, Jr. and C. H. Papadimitriou, *A fast algorithm for testing for safety and deadlocks in locked transaction systems*, J. Algorithms, 2 (1981), pp. 211–226.

[Pa1] C. H. Papadimitriou, *Serializability of concurrent updates*, J. Assoc. Comput. Mach., 26 (1979), pp. 631–653.

[Pa2] ———, *The Theory of Database Concurrency Control*, monograph, in preparation.

[Pa3] ———, *On the power of locking*, Proc. 1981 SIGMOD Conference

[PBR] C. H. Papadimitriou, P. A. Bernstein and J. B. Rothnie, *Computational problems related to database concurrency control*, Conference on Theoretical Computer Science, University of Waterloo, Ontario, 1977, pp. 275–282.

[SK] A. Silberchatz and Z. Kedem, *Consistency in hierarchical database systems*, J. Assoc. Comput. Mach., 27 (1980), pp. 72–80.

[SLR] R. E. Stearns, P. M. Lewis and D. J. Rosenkrantz, *Concurrency control for database systems*, in Proc. 17th Annual Symposium on Foundations of Computer Science, 1976, pp. 19–32.

[Ya] M. Yannakakis, *A theory of safe locking policies in database systems*, J. Assoc. Comput. Mach., to appear.

[YPK] M. Yannakakis, C. H. Papadimitriou and H. T. Kung, *Locking policies: Safety and freedom from deadlock*, Proc. 20th ACM Conference on Foundations of Computer Science, 1979, pp. 283–287.