

## Concurrency Control in Graph Protocols by Using Edge Locks\*

Gael N. Buckley  
Avi Silberschatz

Department of Computer Science  
University of Texas at Austin  
Austin, Texas 78712

### Abstract

A large number of locking protocols use precedence relations among data items to ensure the serializability of the database system. These protocols have extended the semantics of the exclusive lock from prohibiting access to a data item to prohibiting access to an entire subgraph. In this paper we argue that combining the use of exclusive locks for these different purposes is ill conceived. We present a general theory on how these two distinct functions can be separated into the traditional locks operating on the individual data items, and a corresponding set operating on the edges of graph. This is illustrated by a general transformation from a given graph protocol to the new edge protocol which preserves the major properties of the original protocol. We then give a characterization for a large class of edge lock protocols within a database system, which includes most previous locking protocols defined on databases organized as graphs. We show how this separation increases concurrency, and generalizes previously unrelated concepts, such as using deadlock avoidance locks in general locking protocols.

### 1. Introduction

A database system has a set of user programs  $P$  which can operate on the set of data items  $D$ . If it is assumed that each user program (or transaction) when executed alone preserves the consistency of the database, then it is often necessary to ensure that any concurrent execution of several user programs also preserves consistency of the database. A database system that guarantees this property is said to be *serializable* [2].

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

Most systems ensure serializability by controlling access to each item in the set  $D$  via the use of a concurrency control scheme. A very common model for such systems is the use of a *locking protocol*. A transaction must lock a data item before the first access to that item, and unlock it when all accesses to the item are complete. Locking protocols commonly use two types of locks, *exclusive* locks and *shared* locks. Exclusive locks preclude any other lock to be held on a data item, while shared locks permit other shared locks on the same item. Thus a locking protocol is a restriction on when a transaction may lock and unlock each of the data items in the database.

Eswaran et al [2], were the first to discuss the importance of serializability. They also introduced the first useful locking protocol - the *two phase* protocol (denoted 2PL). This protocol specifies that no data item can be unlocked until all data items to be accessed have been locked. Eswaran et al have also demonstrated that if no additional information concerning the structure of the database is available, the two phase criterion is necessary and sufficient to ensure serializability. To overcome this problem, several authors have used a precedence order on the database structure to develop locking protocols that are not two phase [7, 4, 3, 8]. We term these *graph protocols*.

The earliest of the graph protocols is the tree[X] protocol [7], used in databases organized (logically or physically) as trees. In this protocol, a transaction can lock any data item in exclusive mode first, and lock a subsequent item in exclusive mode only if its father is locked, and the item has not yet been locked. A transaction can unlock a data item at any time. In this protocol exclusive locks are used for three different

\*This research was supported in part by the Office of Naval Research under Contract N00014-80-K-0987, and by the National Science Foundation under Grant MCS 81-04017.

purposes. These are:

- to prohibit access of other transactions from its unlocked data items to the subtree where it will yet lock additional data items,
- to prohibit access to an individual data item that it intends to update, and
- to prevent deadlock.

Hence, the tree protocol extended the semantics of the exclusive lock.

The idea of restricting access to parts of the database graph was further developed by more general non-two-phase graph protocols, applicable to both acyclic and arbitrary directed graphs (see, for example [8, 4, 3]). These protocols operate on a database modelled as a directed graph or hypergraph. In general, the restrictions for these protocols state that an item cannot be locked unless the transaction is holding locks on a set of items (usually some number of fathers of the item), and that this set has some functional relation to other sets of the item in order to maintain serializability (and perhaps deadlock freedom).

This paper will show that the idea of using exclusive locks on data items to prohibit access to the rest of the graph is ill conceived, and will elaborate on some problems which arise with the use of such methods. We show that these problems may be eliminated by the use of an *edge lock*, which operates on the precedence relations between data items rather than the data items themselves. In this manner the subgraph exclusion of the previous protocols is clearly separated from exclusion to an individual data item, and thereby enhances the concurrency of the original protocols. Edge locks were originally introduced by Korth to prevent deadlock in databases that allowed multiple granularity locks [6]. We begin with a simple transformation from existing graph protocols to the corresponding edge protocols, and then develop a more general theory for edge locks.

The remainder of this paper is organized as follows. Section 2 describes the model. Section 3 motivates the use of edge locks by illustrating how concurrency is enhanced by using the edge analogue of the family of tree protocols. Section 4 presents the general transformation between a graph protocol and its corresponding edge protocol. This transformation shows that the two phase protocol operating in rooted acyclic graphs is a member of the edge guard[XS] protocol [4]. In section 5 a general characterization is given for a class of edge locks protocols which include most previous locking protocols defined on graphs. We conclude the paper with a discussion on how deadlock avoidance locks on edges (introduced by Korth [6]) play an important part in general graph protocols.

## 2. The Edge Lock Model

It is assumed that the set  $D$  is organized in the form of an arbitrary hypergraph consisting of data items and *hyperedges*. An undirected hyperedge is a set of data items. It is used to define the access relations between data items, where a data item  $A$  can be accessed if and only if every other item in some hyperedge that includes  $A$  has previously been accessed. The use of hyperedges rather than simple edges allows easier specification of the access restrictions imposed on the database. Most locking protocols operate on directed hypergraphs, where each hyperedge has at least one data item in its tail, and exactly one data item as its head. A transaction which will access some data item  $A$  for the first time must have previously accessed all data items in the tail of some hyperedge for which  $A$  is the head (denoted an *incoming edge* of  $A$ ). This restriction does not apply to the first data item accessed by a transaction. If each data item can be accessed independently,  $D$  is to be considered a complete graph.

**Definition 2-1:** A *path* exists from item  $A$  to item  $B$  if  $A$  and  $B$  are members of the same hyperedge, or  $B$  and  $C$  are members of the same hyperedge and there is a path from  $A$  to  $C$ .

**Definition 2-2:** A set  $Q$  of items *separates* item  $A$  from item  $B$  if deleting all hyperedges containing an element of  $Q$  removes every path from  $A$  to  $B$ . A set of hyperedges *separates*  $A$  from  $B$  if deleting all hyperedges removes every path from  $A$  to  $B$ .

A transaction accesses data items according to the restrictions imposed by the hyperedges defined in  $D$ , locking an item in mode  $M$  before the first access to that item, and releasing the lock after all accesses to the item are complete. In this paper we consider shared and exclusive lock modes on data items (denoted  $S$  and  $X$ , respectively), where a transaction is restricted to reading the value of item  $A$  while  $A$  is locked in  $S$  mode, and can read or update  $A$  while  $A$  is locked in  $X$  mode. A lock request in  $S$  or  $X$  mode *conflicts* with a lock issued in  $X$  mode, and hence an  $X$  lock on item  $A$  precludes any other lock on item  $A$ . Locks in  $S$  mode do not conflict with  $S$  mode, which implies several  $S$  locks may be held simultaneously on item  $A$ . A lock or unlock instruction on item  $A$  is denoted by  $LS(A)$ ,  $LX(A)$  or  $UN(A)$ .

**Definition 2-3:** A history  $H$  is the chronological sequence of the lock and unlock operations of an arbitrary set of transactions  $T = \{T_0, \dots, T_{N-1}\}$ . We define the *less than* relation  $<$  on a history  $H$  of a set  $T$  of transactions as follows:

$T_i < T_j \Leftrightarrow T_i$  and  $T_j$  lock item  $A$  in conflicting mode, and  $T_i$  locks  $A$  before  $T_j$ .

**Lemma 2-1:** A protocol ensures serializability if and only if all possible executions for a set of transactions produces an acyclic  $<$  relation.  $\square$

A transaction also issues locks on edges, which can be in one of several modes. Since we restrict the data item lock modes to S and X, all protocols with only one class of transactions need only two edge lock modes. In a later section we introduce protocols which have separate classes of transactions, and which use more than two edge lock modes. The first lock mode is denoted EX, and prevents transactions that come after  $T_i$  in the history H to enter the subgraph that  $T_i$  is operating in. The second mode allows transactions which do not come after  $T_i$  in H to enter the subgraph, and is denoted ES. Locks in EX mode conflict with EX and ES, which implies an EX lock precludes other locks on an edge. Several ES locks may be held concurrently on an edge. The issue of a lock or unlock on edge  $\langle A, B \rangle$  is denoted by LEX( $\langle A, B \rangle$ ), LES( $\langle A, B \rangle$ ), or UN( $\langle A, B \rangle$ ).

If transaction  $T_i$  holds a lock on a data item or edge in mode M and  $T_j$  requests a lock in conflicting mode on the same entity, then  $T_j$  must wait for  $T_i$ . Deadlock occurs when a set of transactions creates a wait for cycle. For the locking protocols discussed throughout this paper, rollback occurs when some transaction in deadlock must be removed to enable further progress of the system.

### 3. Transformation from a Graph Protocol to its Corresponding Edge Protocol

In this section we give a simple transformation that takes any graph protocol defined on directed hypergraphs (and thus regular graphs) and produces an edge protocol which is serializable (and deadlock free) if the graph protocol was serializable (and deadlock free). The corresponding edge protocol allows a larger set of serializable histories than the graph protocol, as is illustrated in the following example. We present the original graph protocol, its corresponding edge protocol, the graph protocol history, and the edge lock history that illustrates the additional concurrency gained by edge locks.

We begin with the tree[X] protocol [7], which requires a transaction to issue only X locks. In the tree[X] protocol,  $T_i$  can request a lock on vertex A iff:

- A is the first vertex locked by  $T_i$ , or the father of A is locked in X mode by  $T_i$
- A has not yet been locked by  $T_i$ .

A vertex can be unlocked at any time.

The corresponding edge lock protocol derived from the previous protocol is stated as follows: A transaction  $T_i$  can lock edge  $\langle A, B \rangle$  in EX mode iff:

- $\langle A, B \rangle$  is the first edge to be locked by  $T_i$ , or  $T_i$  has a lock on the father of  $\langle A, B \rangle$
- $\langle A, B \rangle$  has not been previously locked by  $T_i$ .

$T_i$  can lock an item A in X mode if it holds a lock on the incoming edge  $\langle Q, A \rangle$ , where Q is the father of item A.

The original tree protocol unnecessarily restricts access to a subgraph reachable from a data item while the data item is being updated. The following partial history illustrates this, where two transactions  $T_1$  and  $T_2$  operate in the database of figure 1.  $T_1$  locks data items A and C for 10 time units each, and  $T_2$  locks item B for 100 units.

$T_1$	$T_2$
LX(A)	
	LX(B)
LX(B)	UN(B)
LX(C)	

In this example,  $T_1$ 's completion time (or response time) is 120 units, even though the two transactions lock no data items in common. All graph protocols that use only exclusive locks and operate on the same tree have sets of transactions with identical behavior. Note that  $T_1$ 's completion time for the 2PL protocol is only 20 units, a large performance difference, especially to a database user.

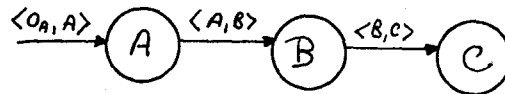


Figure 1

The use of edge locks removes this restriction, as the following history for transactions  $T_1$  and  $T_2$  illustrates. For uniformity of presentation, a transaction can only lock a root A of a graph when it holds a lock on a dummy incoming edge, termed  $\langle 0_A, A \rangle$ .

$T_1$	$T_2$
LEX( $\langle 0_A, A \rangle$ )	
LX(A)	LEX( $\langle A, B \rangle$ )
	LX(B)
LEX( $\langle A, B \rangle$ )	UNE( $\langle A, B \rangle$ )
LEX( $\langle B, C \rangle$ )	
LX(C)	
	UN(B)

This example illustrates how the use of edge locks allows  $T_1$  to bypass  $T_2$ , and drastically decreases its completion (or response) time.

#### 4. The General Transformation

We now present the general transformation from a graph protocol to its corresponding edge protocol, such that all major properties are preserved. To state it succinctly, we need to add some virtual edges to the original graph.

In a hypergraph, if there is a path of length  $M$  from item  $A$  to item  $B$ , there is a path of length  $M$  from an incoming edge of  $A$  to an incoming edge of  $B$  (by induction on the length of the path). Hence, it follows that all terms such as father, ancestor, etc. can be used in an equivalent sense when describing the edge protocol. To the original graph, we add an incoming edge  $\langle O_A, A \rangle$  for each root  $A$  in the graph, and a dummy outgoing edge  $\langle A, I_A \rangle$  for every vertex in the graph. (This second dummy edge is used only to make the statement of the transformation succinct.)

The transformation is as follows.

- Replace every reference to vertex  $A$  with a reference to edge  $\langle Q, R \rangle$ , and every reference to  $B$  with  $\langle S, T \rangle$  such that the graph relations between edges  $\langle Q, R \rangle$  and  $\langle S, T \rangle$ , are identical to the relations between their replaced vertices  $A$  and  $B$ . It is sufficient but not necessary that  $\langle Q, R \rangle$  and  $\langle S, T \rangle$  are incoming edges to  $A$  and  $B$ .
- $T_i$  can request a lock in mode  $M$  on item  $A$  iff  $T_i$  can request a lock in mode  $EM$  on edge  $\langle A, I_A \rangle$ . The first entity locked by  $T_i$  must be an edge.
- $T_i$  can unlock vertex  $A$  only when  $T_i$  can unlock  $\langle A, I_A \rangle$  without invalidating any future edge lock requests necessary to lock additional data items, as defined by the edge protocol above.

**Theorem 4-1:** If the original graph protocol is serializable, then the corresponding edge protocol is serializable.  $\square$

**Theorem 4-2:** If the original graph protocol is deadlock free (or allows simple deadlock), then the corresponding edge protocol is deadlock free (or allows simple deadlock).  $\square$

To illustrate this transformation we use the tree[XS] protocol [4], where a transaction can issue both X and S mode locks on data items. This edge protocol is especially interesting since it includes the two phase protocol operating in trees. The statement of the original protocol uses the following definition:

- the value of function  $\text{top}(A, T_i)$  is the son of the closest ancestor of  $A$  locked in X mode if one exists, otherwise it is the first item locked by  $T_i$ .

Using the tree[XS] protocol,  $T_i$  can lock vertex  $A$  in either S or X mode iff:

1. a.  $A$  is the first vertex locked by  $T_i$  or
  - i. all vertices except  $A$  in the path( $\text{top}(A, T_i), A$ ) are locked in S mode by  $T_i$ , and
  - ii. all vertices  $R$  with  $\text{top}(R, T_i) = \text{top}(A, T_i)$  have not yet been unlocked.
  - iii. If  $\text{top}(A, T_i)$  is the first vertex locked by  $T_i$ , then no item has been unlocked.
2.  $V$  has not yet been locked by  $T_i$ .

An example of these conditions are shown in figure 2, where  $T_i$  has locked  $A$  in X mode,  $B, C$ , and  $D$  in S mode, and it wishes to lock  $E$ .  $\text{Top}(E, T_i)$  is  $B$ , and hence  $D$  must be locked in S mode, and  $B, C, D$  must still be locked by  $T_i$ .

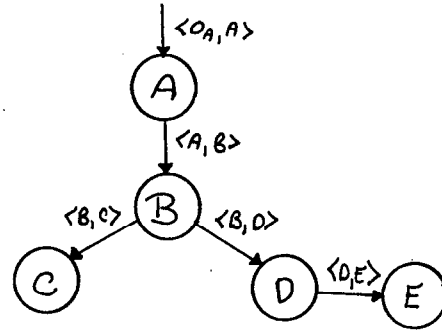


Figure 2

We use the general transformation to change the tree[XS] protocol to the corresponding edge lock protocol. The new protocol is stated using the corresponding definition for  $\text{etop}(\langle M, V \rangle, T_i)$ , where

- the value of function  $\text{etop}$  is the son of the closest ancestor locked in EX mode by  $T_i$ , otherwise it is the first edge locked by  $T_i$ .

$T_i$  can lock edge  $\langle M, V \rangle$  in S or X mode using the tree[XS] edge protocol iff:

1. a.  $\langle M, V \rangle$  is the first edge to be locked by  $T_i$  or
  - i. all edges except  $\langle M, V \rangle$  in the path( $\text{etop}(\langle M, V \rangle, T_i)$ ) are locked in ES mode and
  - ii. all edges  $\langle R, Q \rangle$  with  $\text{etop}(\langle R, Q \rangle, T_i) = \text{etop}(\langle M, V \rangle, T_i)$  are locked by  $T_i$
  - iii. If  $\text{etop}(\langle M, V \rangle, T_i)$  is the first edge locked by  $T_i$ , then no edge has been unlocked,
  - iv. If  $T_i$  has locked the vertex  $Q$  for any edge  $\langle R, Q \rangle$  locked in ES mode fulfilling conditions i or ii above,  $T_i$  still has a lock on  $Q$ .

2.  $\langle M, V \rangle$  has not been locked by  $T_i$ .

$T_i$  can lock item A when some  $T_j$  could request a lock on the outgoing dummy edge  $\langle A, I_A \rangle$  using the above protocol.

We now illustrate how two transactions with no data items in common are restricted by the subgraph restriction of X mode locks. Consider again the database of figure 2, where transaction  $T_1$  reads A and C for 10 units, and  $T_2$  writes D and B concurrently for 50 units total. Consider the following history:

$T_1$	$T_2$
	LX(B)
	LX(D)
LS(A)	
	UN(B)
LS(B)	
LS(C)	

In this case,  $T_1$  must wait 30 additional units until  $T_2$  can unlock B.

If  $T_1$  and  $T_2$  were to execute under the corresponding edge lock protocol, they could produce the same history as the two phase protocol. We now give the corresponding edge history for the two transactions above.

$T_1$	$T_2$
	LES( $\langle A, B \rangle$ )
	LES( $\langle B, D \rangle$ )
LES( $\langle O_A, A \rangle$ )	
LES( $\langle A, B \rangle$ )	
LES( $\langle B, C \rangle$ )	
	LX(B)
	LX(D)
LS(A)	
LS(C)	

Notice that all edges can be shared as long as no transaction has unlocked a data item. The original tree[XS] protocol allowed simple deadlock, as does the corresponding edge protocol.

**Corollary 4-1:** The edge tree[XS] protocol admits all two phase histories that operate on trees and allow only simple rollback.  $\square$

## 5. A Theory for Edge Locks with ES and EX modes

In this section we present a characterization for a large class of locking policies that have been defined for structured databases. It incorporates the use of ES and EX locks, and also information about the behavior of transactions in the database system.

If a transaction  $T_i$  in the set  $T$  is an L policy (Yannakakis [8]), it can only use information about the

structure of the database and any previous locks it has issued to determine which items it is still able to lock. We use the same definition for edge lock L policies. It was shown in [8] that a transaction which issues X locks only must have a lock on all the items in the tail of a hyperedge and the items presently locked in X mode must separate all unlocked items from all items yet to be locked in order to ensure serializability.

For transaction  $T_i$  to issue a lock request on edge  $\langle R, Q \rangle$ , it must hold an edge lock on an incoming edge for some item in the tail of edge  $\langle R, Q \rangle$ . For transaction  $T_i$  to issue a lock request on item A, it must hold a lock on an incoming edge to A. This behavior holds for both the L policies and CLASSL policies described below.

**Theorem 5-1:** Any L policy using edge locks in ES and EX mode and data item locks in X mode is serializable iff

- When transaction  $T_i$  issues a lock request on item A, the edges locked in EX mode separate item A from any item that  $T_i$  has unlocked.

$\square$

Notice that this use of edge locks further extends the allowable histories for transactions that issue only X and EX locks, since the lock on the hyperedge to A may be held in ES mode, and may be used as a path for the transactions not blocked by the edges held by  $T_i$  in EX mode.

**Corollary 5-1:** Edge lock L policies recognize a larger set of serializable histories than L policies using only data item locks.  $\square$

We now present the final class of graph policies covered in this paper, those that, in addition to using the information available to L policies, divides the set  $T$  into distinct classes, where each class restricts transaction behavior in a proscribed manner. We term these CLASSL policies. These policies include the guard[X,S] policy [4], in which a transaction issues either only X locks or only S locks. That protocol requires a transaction which issues X locks to begin at the root of a rooted acyclic graph.

Since this paper contains protocols which use only S and X locks, we discuss CLASSL policies where transactions in class  $C_1$  are allowed to issue only S locks, and those in class  $C_2$  can issue both S and X locks. Transactions in class  $C_1$  must restrict the access of any transaction in class  $C_2$  by using the L policy defined earlier. However, it is not necessary to exclude other members of  $C_1$  in certain areas of the graph. To take advantage of this, the two classes of transactions should have distinct types of locks.

We define two new locks, ES1 and EX1, which have the following compatibility relation.

	ES	ES1	EX	EX1	
ES	o	o	x	o	
ES1	o	o	x	x	o-compatible
EX	x	x	x	x	x-conflicting
EX1	o	x	x	o	

Transactions in class  $C_1$  issue ES, EX, and EX1 locks, and transactions in class  $C_2$  issue ES1 and EX locks. This separation of shared edge locks is necessary so that transactions in class  $C_2$  can be easily distinguished from transactions in class  $C_1$ .

We define the  $C_1$ - $C_2$  CLASSL policy as follows. All transactions in class  $C_2$  follow the L policy described above. A transaction  $T_i$  in class  $C_1$  must satisfy one of the following conditions:

- If  $T_i$  issues a lock on item A, has unlocked a data item B which can be accessed by a transaction of class  $C_2$ , and there is a path between A and data items accessible by a transaction of class  $C_2$  which need not lock B,  $T_i$  must follow the L edge lock policy defined above.
- Otherwise,  $T_i$  must have separated all unlocked items from items yet to be locked by a set of edges locked in EX1 mode.

**Theorem 5-2:** The  $C_1$ - $C_2$  CLASSL policy is serializable.  $\square$

This protocol illustrates the applicability of the edge lock model to capture the behavior of quite specialized locking protocols. The new lock modes clearly illustrate the restrictions on the allowed access paths for transactions of both classes. A general theory of arbitrary lock-modes is given in [5].

## 6. Conclusion

We have introduced a new method of separating the several functions that exclusive and shared locks represent in non-two-phase graph protocols. We have shown that there exists simple transformations from existing graph protocols to their corresponding edge protocols which admits the same behavior as the original graph protocol. We have given a theory on the use of ES and EX locks in a structured database which admits of and extends the serializable histories for many previously defined graph protocols.

We discuss one related concept, how deadlock avoidance edges (as defined for variable granularity locks by Korth [6]) can be used in general edge graph protocols. The general application of the L policy to a tree structured database can result in arbitrary deadlocks, thereby causing transactions to be rolled back. In many protocols the use of X locks (and the use of EX edge locks in an analogous manner), prevented this performance liability from occurring.

We indicate how deadlock avoidance edges would be used to prohibit such histories. If a transaction begins at one item in the tree and uses ES mode edge locks to access needed data items, it will need an additional deadlock freedom lock that specifies whether data items that can be accessed from this edge could be locked in X mode (termed SOMEX), or will all be locked in S mode (termed ALLS). Locks in ALLS mode are compatible with other locks in ALLS mode, but locks in SOMEX mode must be incompatible with ALLS and SOMEX in order to prevent deadlock. This technique can be used only when the graph structure is sufficiently restricted to ensure deadlock freedom (see [1, 9]). Again, this extends the set of serializable histories while preserving desired properties.

## References

1. Buckley, G, and Silberschatz, A. Eliminating Cascading Rollback in Structured Databases. The University of Texas at Austin, October, 1983.
2. Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, I.L. "The Notions of Consistency and Predicate Locks in a Database System." *Communications of the ACM* 10, 11 (Nov. 1976), 624-723.
3. Kedem, Z., Silberschatz, A. Non-Two-Phase Locking Protocols with Shared and Exclusive Locks. Proc. Sixth International Conference on Very Large Data Bases, October, 1980.
4. Kedem, Z. and Silberschatz, A. "Locking protocols: from Exclusive to Shared Locks." *Journal of the ACM* 30, 4 (October 1983), 787-804.
5. Korth, H. "Locking Primitives in a Database System." *Journal of the ACM* 30, 1 (January 1983), 55-79.
6. Korth, H. "Deadlock Freedom Using Edge Locks." *ACM Transactions on Database Systems* 7, 4 (December 1982), 632-652.
7. Silberschatz, A., Kedem, Z. "Consistency in Hierarchical Database Systems." *Journal of the ACM* 27, 1 (Jan. 1980), 72-80.
8. Yannakakis, M. "A Theory of Safe Locking Policies in Database Systems." *Journal of the ACM* 29, 4 (July 1982), 221-244.
9. Yannakakis, M. "Deadlock in Locking Policies." *Siam Journal of Computing* 11, 2 (May 1982), 391-408.