GIT-ICS-81/05

Multilevel Atomicity - A New Correctness Criterion For Distributed Databases *

Nancy A. Lynch

May 1981

*This research was supported in part by the NSF under Grant No. MCS79-24370, U.S. Army Research Office Contract #DAAG29-79-C-0155 and Office of Naval Research Contract #N00014-79-C-0873.

MULTILEVEL ATOMICITY - A NEW CORRECTNESS CRITERION FOR DISTRIBUTED DATABASES *

Nancy A. Lynch
Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

May 1981

*This research was supported in part by the NSF under Grant No. MCS79-24370, U.S. Army Research Office Contract #DAAG29-79-C-0155 and Office of Naval Research Contract #N00014-79-C-0873.

CONTENTS

I. Introduction

The usual models for distributed databases [RSL,BG] are based on a set of "entities" distributed among the nodes of a network. These entities are accessed by users of the database through "transactions", which are certain sequences of steps ("actions") involving the individual entities. The steps are grouped into transactions for two distinct purposes. First, a transaction is used as a unit of recovery: either all of the steps of a transaction should be carried out, or none of them should; thus, if a transaction cannot be completed, its initial steps must be "undone" in some way. Second, a transaction is used to define atomicity: all of the steps of a transaction form a logical atomic unit in the sense that it should appear to users of the database that all of these steps are carried out consecutively, without any intervening steps of other transactions. This requirement that transactions appear to be atomic is called "serializability" in the literature [EGLT,RSL,BG], and has been widely accepted as an important correctness criterion for distributed databases.

It seems to me that these two purposes should not be served by the same transaction mechanism. While I think the usual notion of "transaction" is adequate for purposes of recovery, I think that it is less appropriate for defining atomicity. Namely, the requirement of serializability is so strong that it seems to exclude efficient implementation of many application databases. This paper suggests superimposing a new mechanism on the transaction mechanism, in order to define atomicity.

The model I use in this paper for a distributed database consists of two completel distinct levels - a physical level consisting of node processors connected by a message system and communicating with users by ports, and a logical level consisting of a <u>centralized concurrent</u> application database. (The logical level does not involve nodes, messages, or any other distribution information.) It is the job of the physical system to "implement", in some appropriate sense, the application database.

The steps of different application database transactions might be allowed to interleave in various ways; the set of allowable interleavings is determined by the application represented. At one extreme, it might be specified that all allowable interleavings be serializable; this amounts to requiring that the application database be a <u>centralized serial</u> database. At the other extreme, the interleavings might be unconstrained. In a banking database, a transfer transaction might consist of a withdrawal step followed by a deposit step. In order to obtain fast performance, the withdrawals and deposits of different transfers might be allowed to interleave arbitrarily, even though the users of the banking database are thereby presented with a view of the account balances which includes the possibility of money being "in transit" from one account to another. In between the two extremes, there are many other reasonable possibilities.

In [FGL], we assume an application database allowing any set of allowable interleavings of transactions. We show how to modify a distributed system implementing such an application, so that it has an additional capacity to determine a global database state, without stopping transactions in progress. Consistency of such a global database state can be checked, and repeated use of this capacity can also aid in recovery from inconsistent global states. In that work, any set of allowable interleavings can be assumed; we guarantee that if the original distributed system only produces allowable interleavings, then the modified system will also produce only allowable interleavings. Thus, a global state can be obtained for application databases which are serializable, arbitrarily interleaved, or anything in between these two extremes.

In this paper, only certain sets of interleavings are considered. The intention is to consider only sets of interleavings which can be specified in a way which is suitable for use by a concurrency control algorithm. At the same time, the sets of interleavings considered should be general enough to allow representation of the allowable interleavings for important application databases such as those for banking.

As a first approximation to a specification method, we might associate with each transaction its "atomicity" (or "granularity" [GLPT]), which is formally described by a set of "breakpoints" between different sets of consecutive steps. Steps not separated by a breakpoint would always be required to occur atomically, (at least from the point of view of the system users). As a special case of this definition, if there are no breakpoints for any transaction except at the beginning and end, then this requirement is simply the usual requirement of serializability. As another special case, if there are always breakpoints between every pair of steps of each transaction, then this requirement allows arbitrary interleaving. In addition, many intermediate cases are possible.

However, this definition does not seem to me to be sufficiently general to express all commonly-used constraints on interleavings. For example, consider a banking system with transfer transactions as described above. Transfers might be allowed to interleave arbitrarily with each other. However, we might also want to have another type of transaction, an "audit transaction" [FGL], which reads all of the account balances and returns their total. This audit transaction should probably not be allowed to interrupt a transfer transaction between the withdrawal and deposit steps, for then the audit would miss counting the money "in transit". That is, the entire transfer transaction should be atomic with respect to the entire audit transaction. Thus, the same transfer transaction should have one set of breakpoints with respect to other transfers, and another set with respect to audit transactions.

This example seems to be representative of a fairly general phenomenon: it might be appropriate for a transaction to have different sets of breakpoints with respect to different other transactions. That is, each transaction might allow different "views" of its activity to different other transactions. Thus, a natural specification for allowable interleavings might be in terms of the "relative atomicity" of each transaction with respect to each other transaction, rather than just in terms of each transaction's (absolute) "atomicity".

In this paper, a formal definition is given for a type of relative atomicity, called "multilevel atomicity". The two-level model for distributed databases is described. A combinatorial lemma is presented, which yields a necessary and sufficient condition for achieving multilevel atomicity. Some suggestions are made for using this condition as the basis for a concurrency control design for multilevel atomicity.

Other researchers [L,GLPT,G,C] have also noted that the usual notion of serializability needs to be weakened. In particular, [G] contains interesting preliminary work on specification and concurrency control design, for certain non-serializable interleavings. The multilevel atomicity of this paper is a generalization of the two-level atomicity described in [G] under the designation "compatibility sets".

Much work remains to be done, in designing and evaluating concurrency control algorithms for multilevel atomicity. This paper merely suggests some preliminary definitions and ways in which they might be used. It remains to see whether new concurrency control algorithms which achieve multilevel atomicity can be made to operate much more efficiently than existing concurrency control algorithms which achieve serializability. It also remains to determine whether these weaker notions than serializability are useful for describing the constraints required for real-world database applications.

2. A Model for Asynchronous Parallel Processes

Both the application databases and the physical systems of this paper can be formalized within the model of [LF] for asynchronous parallel computation. This unified model allows precise description of distributed algorithms as processes accessing variables (i.e. either shared variables or distributed system message ports). In this paper, I will be informal. Only a brief description of the model is provided; the reader is referred to [LF] for a complete, rigorous treatment.

The basic entities of the model are processes (nondeterministic automata) and variables. Processes have states (including start states and possibly also final states), while variables take on values. An atomic execution step of a process involves accessing one variable and possibly changing the process' state or the variable's value or both. A system of processes is a set of processes, with certain of its variables designated as internal and others as external. Internal variables are to be used only by the given system. External variables are assumed to be accessible to some "environment" (e.g. other processes or users) which can change the values between steps of the given system.

The execution of a system of processes is described by a set of execution sequences. Each sequence is a (finite or infinite) list of steps which the system could perform when interleaved with appropriate actions by the environment. Each sequence is obtained by interleaving sequences of steps of the processes of the system. Each process must have infinitely many steps in the sequence unless that process reaches a final state.

For describing the external behavior of a system, certain information in the execution sequences is irrelevant. The external behavior of a system of processes is the set of sequences derived from the execution sequences by "erasing" information about process identity, change of process state and accesses to internal variables. What remains is just the history of accesses to external variables.

A distributed problem is any set of sequences of accesses to variables. A system is said to solve the problem if its external behavior is any subset of the given problem.

In this paper, the technical assumption that no state can be both a start state and a final state is required. Also, one general definition not in [LF] is required. Namely, if S and S' are systems, then S is a subsystem of S' if the processes, internal variables and external variables of S are included, respectively, among those of S', and the internal variables of S are initialized exactly as they are in S'.

3. Application Databases

My notion of an application database is a centralized, concurrent system consisting of transactions acting on entities, together with a set of allowable interleavings of the steps of those transactions. This is modelled very directly in the model of Section 2: transactions are simply formalized as processes, while entities are formalized as variables. More precisely, an application database (S,A) consists of a system S of processes (called transactions), together with a subset A of the execution sequences of subsystems of S (called the allowable execution sequences), such that the following two conditions are satisfied.

(a) All variables of S are internal (i.e. internal to the system). (They are called entities. This assumption says that the entities are only accessed via the transactions.)

(b) In every execution sequence e in A, every transaction which appears, eventually appears in a final state. (Thus, all transactions are supposed to terminate.)

This definition gives a very general notion of an application database. The (indivisible) steps of transactions are arbitrary accesses to entities, not necessarily just reading or writing steps (although these two types of steps are permissible special cases). Transactions can branch conditionally: for example, based on the values encountered for certain entities, they might access different entities at later steps. This model of a transaction is general enough to include most others in the literature. It also includes some other notions usually regarded as somewhat different from ordinary transactions: the "transactions with revoking actions" in [G] are a particular type of nondeterministic transaction in the present model.

4. Coherent Partial Orders

I want to show how to describe certain sets of allowable execution sequences. In this section, I present some preliminary, rather abstract, definitions involving sets and partial orders. The definitions of this section are given at an abstract level since they will be used for a general combinatorial lemma in Section 7.

I first describe the partitions of an arbitrary set T (to be thought of as a set of transactions) into levels.

A k-nest, $\Pi = (\pi_1,\ldots,\pi_k)$ for a set T is a sequence of equivalence relations on T, satisfying the following conditions:

(a) π_1 consists of exactly one equivalence class,

(b) π_k consists of singleton equivalence classes, and

(c) Each π_i is a refinement of its predecessor, π_{i-1}.

If $t, t' \in T$, then $\text{level}_\Pi(t,t')$ is the largest i for which $t\ \pi_i\ t'$.

Next, I describe an abstract "breakpoint" function which defines a set of breakpoints within a totally ordered set for each of several "levels", in such a way that the higher level sets of breakpoints always include the lower level sets. Each totally ordered set should be thought of as the set of steps of some execution sequence of a particular transaction.

If X is totally ordered by $\le$, $k \in \mathbb{N}$, then a k-level breakpoint function, b, for $(X,\le)$ assigns a set of pairs of $\le$-consecutive elements of X to each i, $1 \le i \le k$, in such a way that:

(a) b(1) contains no pairs,

(b) b(k) contains all pairs, and

(c) $b(i) \subseteq b(i+1)$, for all i.

If T is a set, then a <u>k-level interleaving specification</u>, I, for T has the following components:

(a) a collection of disjoint totally ordered sets, $(X_t, \le_t)$, one for each $t \in T$, and

(b) a collection of k-level breakpoint functions, b_t, one for each $(X_t, \le_t)$.

Next, I define an important condition for a partial order on $\bigcup_{t \in T} X_t$. I want to express the fact that $\le$ preserves all of the individual $\le_t$ orderings and also respects the restrictions expressed by the given collection of breakpoint functions.

Let Π be a k-nest for a set including T, $I = \{((X_t, \le_t), b_t) : t \in T\}$ a k-level interleaving specification for T, $\le$ a partial order on $\bigcup_{t \in T} X_t$. Then $\le$ is <u>coherent</u> (for Π and I) provided the following two conditions hold.

(a) The partial order $\le$ contains each partial order $\le_t$.

(b) Assume $\text{level}_\Pi(t,t') = i$. Assume α, $\alpha' \in X_t$ and $\alpha \le_t \alpha'$. Assume $\beta \in X_{t'}$ and $\alpha \le \beta$. Finally, assume there is no pair $(\gamma, \gamma') \in b_t(i)$ with $\alpha \le_t \gamma$ and $\gamma' \le_t \alpha'$. Then $\alpha' \le \beta$.

Intuitively, this latter condition says the following. If a step, β, of one transaction follows a step, α, of another transaction, t, then β also follows any other step, α', of t which follows α but precedes any breakpoints. (Here, "follows" means follows in the partial order $\leq$".) The breakpoints are defined solely by the nesting level i for the two transactions, t and t'.

5. Multilevel Atomicity

The definitions of this section deal explicitly with a system S of transactions. I use the abstract definitions in the preceding section to help describe sets of allowable execution sequences. Intuitively, transactions are grouped in nested classes so that for each t, the set of places where a transaction t' can interrupt t is determined solely by the smallest class containing both t and t'. Moreover, smaller classes determine at least all of the breakpoints determined by containing classes (and possibly more). This says that transactions which are grouped in a common small class might have many relative breakpoints (i.e. can interleave a great deal), while transactions which are only grouped in a common large class might have fewer relative breakpoints (i.e. cannot interleave very much).

For each pair of transactions t and t', I must describe the places at which t is permitted to be interrupted by steps of t'. Since the transactions need not be straight-line programs, but can branch in complicated ways, I am forced to describe separately the places at which each different execution sequence, e, of t can be interrupted by steps of t'.

A <u>k-level breakpoint specification</u>, $\mathcal{B}$, for a system, S, of transactions is a family, $\{b_{t,e}$: t is a transaction of S, e an execution sequence of t$\}$, where each $b_{t,e}$ is a k-level breakpoint function for the steps of e, totally ordered according to their occurrence in e. (Formally, the elements of the ordered set of steps are pairs (i,ξ_i), where ξ_i is the ith step of e.)

A k-nest, Π, for the transactions of a system S and a k-level breakpoint specification, $\mathcal{B}$, for S can be used in a straightforward way to define an application database, $(S,A(\Pi,\mathcal{B}))$. Namely, let e be an execution sequence of a subsystem of S, (i.e. an execution of some of the transactions of S), and T the set of transactions appearing in e. For each $t \in T$, let e_t denote the execution sequence of t occurring as a subsequence of e, X_t the set of steps of t occurring in e_t, $\leq_t$ the order in which those steps occur in e, and let b_t denote $b_{t,e_t} \in \mathcal{B}$. Let $\leq$ denote the total order on $\bigcup_{t \in T} X_t$ describing the order in which all the steps occur in e. Then e is multilevel atomic (for Π and $\mathcal{B}$) provided $\leq$ is coherent for Π and $\mathcal{I} = \{((X_t,\leq_t),b_t):t \in T\}$. (This definition just says that all the interruptions occur at the given breakpoints.) Let $A(\Pi,\mathcal{B})$ denote the set of execution sequences of S which are multilevel atomic for Π and $\mathcal{B}$.

For example, if $\Pi = (\pi_1,\pi_2)$, and $\mathcal{B}$ is the only possible breakpoint specification (i.e. no pairs for $b_{t,e}$ (1), and all pairs for $b_{t,e}$ (2)), then the multilevel atomic execution sequences are just the usual serial executions.

The reader is referred to [G] for treatment of a special case of our definition corresponding to $\Pi = (\pi_1,\pi_2,\pi_3)$, where $b_{t,e}(2)$ consists of all pairs of consecutive steps, for all t and e. That is, transactions in a common π_2 class can interleave arbitrarily, but transactions not in a common π_2 class must be serialized with respect to each other. The "multilevel" definition of this paper also allows intermediate degrees of interleaving as well as the two extremes represented in [G].

6. Simulation of an Application Database

Having described the logical-level centralized and concurrent application database, I now must describe how this database is to be "implemented" by a distributed system (or any other system). The physical system implements the application database by presenting an external interface to the users which is compatible with allowable executions of the application database. Correctness for the physical system is thus defined entirely in terms of its external behavior. The physical system might produce this behavior by many different methods. For example, it might centralize, distribute or replicate the entities. It might implement each transaction on one processor which communicates with other processors in order to access entities. Alternatively, it might divide up the entities among the nodes of a network, and allow transactions to "migrate" from entity to entity as necessary, executing some of their steps on different processors. It is only the external view which determines correctness.

A definition for implementation follows. Fix an application database (S,A). Define a finite nonempty set of variables called ports, each of which can contain a finite set of transaction status words: a transaction status word is a pair (t,s) where t is a transaction of S and s is either a start state or a final state of t. Let a be a sequence of access to ports, each access tagged by the label "users" or "system" (to indicate who is doing the accessing). Then a is syntactically correct provided, in a, the following conditions hold.

(a) Each port starts out empty, and each successive access to a port begins with the same value left at the end of the preceding access to that port.

(b) The changes of port values are all of the following types. The users can initiate a transaction t at any time by inserting a pair (t,s) into a port, where s is a start state of t. The system can change (t,s) to (t,s'), where s is a start state of t and s' is a final state of t.

(c) Each transaction is initiated at most once.
(This is a technical convenience, assumed for the sake of consistency with the formal model of [LF]. If the same transaction is intended to be run twice, it is simply duplicated.)

(d) Each transaction which is initiated by the users is subsequently completed by the system.

It remains to express the semantic requirement that a provide the users with results "consistent with" an allowable execution sequence of the application database.

Let a be a syntactically correct sequence, e an execution sequence of a subsystem of S. Then a is consistent with e provided exactly the same transactions appear in a and e, with the same start states and same final states. A sequence, a, is correct for the users and system together provided a is syntactically correct and consistent with some e in A.

I need a definition of correctness for the system alone. Informally, a system execution sequence is "correct" iff whenever it is run with a "correct user", the result is correct for the users and system together. In a little more detail, a sequence of accesses to ports is correct for the users provided all changes made are among those allowed for the users in (b) and (c) above. (That is, the users can only initiate transactions, cannot retract a transaction once it is initiated, and cannot initiate the same transaction more than once.) Then a sequence is correct for the system provided that whenever it is interleaved consistently with a correct user sequence (and the steps of the resulting sequence labelled appropriately), the result is correct for the users and system together. (The interested reader is referred to [LF] for a completely formal definition for this interleaving.)

A system of processes S' implements application database (S,A) provided all external behavior sequences of S' are correct for the system.

Thus, I use a weak notion of implementation which simply preserves input-output results. I do not require preservation of ordering of transactions; a transaction t is permitted to complete (at a port) before another transaction t' is initiated (at a port) and yet it might be the case that some of the steps of t' precede some of the steps of t in all execution sequences of the application database consistent with the port behavior.

The weakness of the implementation definition allows some freedom in design of the physical system. In particular, for any execution sequence e of a system S of transactions, a dependency partial order $\underset{e}{\leq}$ of the steps of e is defined as follows. For every pair of steps α, τ in e, let $\alpha \underset{e}{\leq} \tau$ if α precedes τ in e and either (i) α and τ are steps of the same transaction, or (ii) α and τ are steps accessing the same entity. Then every total order of the steps of e consistent with $\underset{e}{\leq}$ is also an execution sequence of S, having the same sequence of values for each entity and the same execution subsequence for each transaction, as e.

Two execution sequences, e and e' of S are <u>equivalent</u> if $\underset{e}{\leq}$ is identical to $\underset{e'}{\leq}$. It follows that if a sequence, a, of port accesses is syntactically correct and consistent with an execution sequence, e, which is equivalent to some e' in A, then a is correct.

<u>Example</u>. If A is the set of "serial" executions of the transaction system, then "equivalence with some e in A" amounts to the usual definition for "serializable" executions. If a physical system guarantees that its port behavior is consistent with a serializable execution sequence, then it is also consistent with a serial execution sequence.

<u>Example</u>. A popular model for distributed databases is the "migrating transaction" model described in [RSL]. In this model, entities of the database reside at nodes of a network of processors, and the transactions migrate from entity to entity as necessary, executing some of their steps on different processors. In more detail, a transaction t, with start state s, originates at a processor o. A message (o,t,s) is sent to the processor owning the entity which t accesses when it is in state s. A processor receiving a message (o,t,s) "performs" the indicated step by changing the value of the entity, updating t's state, and sending a new message (o,t,s'), where s' is the new state. If s' is not a final state, the message is sent to the processor owning the appropriate entity. If s' is a final state, the message is sent back to the originator o. In this way, an execution sequence e of the system of transactions is actually "performed" by the processors. (The total order of the sequence is determined by real clock time.) This execution sequence is constructed to be consistent with the port behavior of the system. It suffices for

external port correctness to insure that the execution sequence e "performed" by the processors is one which is equivalent to some e' in A.

Now consider the case in which A is a set of multilevel atomic sequences; that is, assume that Π is a k-nest for the transactions of S, $\mathcal{B} = \{b_{t,e}$: t is a transaction of S, e an execution sequence of t$\}$ is a k-level breakpoint specification for S, and $A = A(\Pi,\mathcal{B})$. We say that an execution sequence e of S is totally coherent (resp. partially coherent) for Π and $\mathcal{B}$ provided the dependency partial order $\underset{e}{\leq}$ is extendable to a total order (resp. partial order) which is coherent for Π and $\mathcal{I} = \{((X_t,\leq_t),b_t : t \in T\}$, where $e_t = (X_t,\leq_t)$ denotes the execution sequence of t occurring as a subsequence of e, and b_t denotes b_{t,e_t}. By definition, an execution sequence e of S is equivalent to one which is multilevel atomic for Π and $\mathcal{B}$ if and only if e is totally coherent for Π and $\mathcal{B}$. Thus, it suffices to insure that each sequence of port accesses is consistent with some totally coherent execution sequence of S. In particular, if the migrating transaction model is used, it suffices to insure that the execution sequence "performed" by the processors is totally coherent.

Note that "totally coherent" generalizes "serializable" in the same sense that "multilevel atomic" generalizes "serial".

It is not immediately obvious how a concurrency control might insure total coherence. Some help is provided by the lemma in the next section.

7. A Combinatorial Lemma

In this section, I state and prove a combinatorial lemma which will be used in the next section to derive a necessary and sufficient condition for multilevel atomicity. The lemma requires only the abstract definitions in Section 4.

For this section, let T be a fixed set, let $\Pi = (\pi_1, \ldots, \pi_k)$ be a fixed k-nest for a set including T, and let $\mathcal{I} = \{((X_t, \leq_t), b_t) : t \in T\}$ be a fixed k-level interleaving specification for T. Let "coherent" mean "coherent for Π and $\mathcal{I}$", and write "level" for "level_Π".

Lemma. If $\leq$ is a coherent partial order, then there is a coherent total order $\leq'$ which contains $\leq$.

Proof. Let $\leq^{(1)}$ denote $\leq$. A sequence of stages numbered 2,...,k is carried out. Each stage, i, inserts additional pairs into the ordering relation, yielding $\leq^{(i)}$. Then $\leq'$ is defined to be $\leq^{(k)}$. It is shown, inductively on i, $1 \leq i \leq k$, that (a) $\leq^{(i)}$ is a coherent partial order, and (b) if $\alpha \in X_t$, $\beta \in X_{t'}$, and level(t,t') < i, then α and β are $\leq^{(i)}$-comparable. Conditions (a) and (b) are trivially true for i = 1. Conditions (a) and (b) for i = k clearly imply the needed result.

Stage i ($2 \leq i \leq k$).

Partition $X = \bigcup_{t \in T} X_t$ into segments, where each segment S is a maximal subset of some X_t with the property that there are no pairs in $b_t(i-1)$ having both components in S. (That is, each X_t is divided into segments at the breakpoints given by $b_t(i-1)$.)

Define a directed graph G whose nodes are all the segments. G contains an edge from segment S_1 to segment S_2 exactly if there exist $\alpha \in S_1$, $\beta \in S_2$ with $\alpha \leq^{(i-1)} \beta$.

Totally order the strongly connected components of G, $\mathcal{S}_1 \leq \mathcal{S}_2 \leq \cdots$, so that G contains no edges from any segment in $\mathcal{S}_m$ to any segment in $\mathcal{S}_n$, $n < m$. Then define $\leq^{(i)}$ by adding to $\leq^{(i-1)}$ all pairs (α,β), where $\alpha \in S_1 \in \mathcal{S}_m$, $\beta \in S_2 \in \mathcal{S}_n$, and $m < n$, and then taking the transitive closure.

END

I now prove the needed properties (a) and (b) for $\leq^{(i)}$, assuming that they hold for $\leq^{(i-1)}$.

<u>Claim 1</u>. $\leq^{(i)}$ is a partial order.

<u>Proof of Claim 1</u>. There are no edges in $\leq^{(i)}$ from $\alpha \in S_1 \in \mathcal{S}_m$ to $\beta \in S_2 \in \mathcal{S}_n$, where $n < m$. Also, all edges in $\leq^{(i)}$ not in $\leq^{(i-1)}$ go from $\alpha \in S_1 \in \mathcal{S}_m$ to $\beta \in S_2 \in \mathcal{S}_n$, where $m < n$. Thus, there is no cycle in $\leq^{(i)}$ involving a new edge. Since $\leq^{(i-1)}$ is a partial order, there are no cycles in $\leq^{(i)}$. □

<u>Claim 2</u>. $\leq^{(i)}$ is coherent.

<u>Proof of Claim 2</u>. Assume level$(t,t') = j$, $\alpha, \alpha' \in X_t$, and $\alpha \leq_t \alpha'$. Assume $\beta \in X_{t'}$, and $\alpha \leq^{(i)} \beta$. Assume there is no pair $(\gamma,\gamma') \in b_t(j)$ with $\alpha \leq_t \gamma$ and $\gamma' \leq_t \alpha'$. I show that $\alpha' \leq^{(i)} \beta$. The result is trivial if $t = t'$, so assume that $t \neq t'$.

Case 1. $\alpha \leq^{(i-1)} \beta$

Then the coherence of $\leq^{(i-1)}$ implies the needed result.

Case 2. $\alpha \nleq^{(i-1)} \beta$

Then $\alpha \in S_1 \in \mathcal{S}_m$, $\beta \in S_2 \in \mathcal{S}_n$ for some $m < n$.

Since $\alpha \leq^{(i)} \beta$ and $\leq^{(i)}$ contains $\leq^{(i-1)}$, it follows that $\beta \nleq^{(i-1)} \alpha$, so that α and β are $\leq^{(i-1)}$-incomparable. Then property (b) applied to $\leq^{(i-1)}$ implies that j (= level(t,t')) $\geq i - 1$. Then $b_t(i-1) \subseteq b_t(j)$ by the definition of a k-level breakpoint function. But S_1 includes all elements from α up to the next $b_t(i-1)$ breakpoint in X_t; since α and α' have no intervening $b_t(j)$-breakpoints, they also have no intervening $b_t(i-1)$-breakpoints, so that $\alpha' \in S_1$. The definition of $\leq^{(i)}$ then insures the needed result.

$\square$

In the following, a segment S is said to belong to an element $t \in T$ if $S \subseteq X_t$.

Claim 3. For each m, the following holds. If S, S' $\in \mathcal{S}_m$, S belongs to t and S' belongs to t', then $t \; \pi_i \; t'$.

Proof of Claim 3. If not, then some $\mathcal{S}_m$ contains a cycle $S_0, S_1, \ldots, S_\ell = S_0$ of segments such that for each j, $0 \leq j \leq \ell-1$, there exist $\alpha \in S_j$, $\beta \in S_{j+1}$ with $\alpha \leq^{(i-1)} \beta$ and such that two of the segments belong to π_i-inequivalent elements of T.

Let S and S' be two distinct segments in this cycle, belonging to elements t and t' respectively, where (i) $t \not\equiv_i t'$, and (ii) any segment S" following S and preceding S' in the cycle belongs to some t" which is π_i-equivalent to t. Then if α is the last (in the $\le_t$-ordering) element of S and β is the last (in the $\le_{t'}$-ordering) element of S',

we claim that $\alpha \le^{(i-1)} \beta$. This is shown by induction on the number of segments following S and preceding S' in the cycle.

Inductive Step. There exists $\alpha' \in S$ such that $\alpha' \le^{(i-1)} \beta'$, where β' is the last step of the cycle-successor of S. By inductive hypothesis (or trivially, if S' itself is S's cycle successor), it follows that $\beta' \le^{(i-1)} \beta$. Thus, $\alpha' \le^{(i-1)} \beta$. Now, $j = \text{level}(t,t') \le i - 1$, by assumption, so $b_t(j) \subseteq b_t(i-1)$. But α precedes the next $b_t(i-1)$ breakpoint following α', so α also

precedes the next $b_t(j)$ breakpoint following α'. Coherence of $\le^{(i-1)}$ implies that $\alpha \le^{(i-1)} \beta$.

Applying this result repeatedly around the cycle shows that there are two distinct segments, S and S', such that $\alpha \le^{(i-1)} \beta$ and $\beta \le^{(i-1)} \alpha$, where α and β are the last steps of S and S' respectively. But this contradicts the assumption that $\le^{(i-1)}$ is a partial order.

□

<u>Claim 4</u>. If $\alpha \in X_t$, $\beta \in X_{t'}$, and level(t,t') < i, then α and β are $\leq^{(i)}$-comparable.

<u>Proof of Claim 4</u>. By Claim 3, t and t' do not have any segments in the same strongly connected component $\mathcal{S}_m$. Thus, $\alpha \in S_1 \in \mathcal{S}_m$, $\beta \in S_2 \in \mathcal{S}_n$, and $m \neq n$. But then $\leq^{(i)}$ is defined to contain the pair (α,β) if $m < n$, and to contain (β,α) if $n < m$.

□

8. A Necessary and Sufficient Condition for Multilevel Atomicity

The lemma of Section 7 is now used to restate the correctness condition at the end of Section 6. Namely, assume that Π and $\mathcal{B}$ are as at the end of Section 6. Then an execution sequence e is equivalent to one which is multilevel atomic for Π and $\mathcal{B}$ if and only if e is partially coherent for Π and $\mathcal{B}$. Thus, it suffices to insure that each sequence of port accesses is consistent with some partially coherent execution sequence of S. In particular, if the migrating transaction model is used, it suffices to insure that the execution sequence e "performed" by the processors is partially coherent for Π and $\mathcal{B}$. In other words, e must have a dependency partial order which is extendable to a partial order which is coherent for Π and $\mathcal{I}$ (where $\mathcal{I}$ is defined as at the end of Section 6).

9. Concurrency Control for Multilevel Atomicity

In this section, I discuss how a concurrency control mechanism might take advantage of some of the preceding ideas. I want to design concurrency controls which use the correctness conditions stated in Section 8. Specifically, I use the migrating transaction model, and consider how to insure that any execution sequence e "performed" by the processors has a dependency partial order $\underset{e}{\leq}$ which is extendable to a coherent partial order.

It will be necessary to make an additional assumption about a breakpoint specification for the application database (S,A). Namely, in order to be able to determine the locations of breakpoints while the execution sequence e is being performed, it is necessary to assume a "compatibility" condition: if two execution sequences of a transaction share a common prefix $\bar{e}$, then either both execution sequences have a breakpoint immediately after $\bar{e}$, or neither does.

In order to insure extendability of $\underset{e}{\leq}$ to a coherent partial order, consider the "smallest possible" coherent extension of $\underset{e}{\leq}$. This can be defined as follows. Given a set T, a k-nest Π for a set containing T, a k-level interleaving specification $I = \{((X_t,\leq_t),b_t):t\in T\}$ for T, and a partial order $\leq$ on $\bigcup_{t\in T} X_t$ containing all the $\underset{t}{\leq}$, define the coherent closure of $\leq$ (with respect to Π and I) to be the partial order obtained from $\leq$ by closing under condition (b) of the coherence definition. Then it is easy to see that $\underset{e}{\leq}$ is extendable to a coherent partial order if and only if the coherent closure of $\underset{e}{\leq}$ is a partial order.

Assume that the concurrency control generates an execution sequence e of S, and that the concurrency control includes some priority scheme and rollback mechanism to insure that no initiated transaction gets blocked indefinitely. (Such a scheme is not specified here.) I consider how to insure that the coherent closure of $\underset{e}{\leq}$ is a partial order.

One possible strategy is cycle-detection, using the coherent closure of $\underset{e}{\leq}$. Namely, if the concurrency control does not otherwise guarantee that $\underset{e}{\leq}$ is extendable to a coherent partial order, the concurrency control might generate explicitly the edges of the coherent closure of $\underset{e}{\leq}$, and check for cycles. If a cycle is detected, a priority scheme can be used to determine which steps should be rolled back. Presumably, fewer cycles would be detected using the multilevel atomicity definition than if serializability were required, leading to fewer rollbacks.

Another approach is to attempt to guarantee that the coherent closure of $\underset{e}{\leq}$ is a partial order. One way of doing this might be to delay some steps, as follows.

Each step β first gets "scheduled", thereby locking its entity and delaying its transaction. β does not actually get "performed" until it insures the following. (Note that e refers to the order in which steps actually get performed, not the order in which they are scheduled.) If e_β is the initial segment of e ending with step β, and if α is the last step of transaction t which precedes β in the coherent closure of $\underset{e_\beta}{\leq}$, then a breakpoint for β's transaction immediately follows α in the execution sequence prefix of t occurring as a subsequence of e_β. (This can be accomplished by making β wait until suitable breakpoints have been reached, assuming that the concurrency control uses a priority-rollback mechanism for preventing blocking.)

If the property above is guaranteed, for each β, then the coherent closure of $\underset{e}{\leq}$ is consistent with the total ordering of steps in e, so it must be a partial order.

Of course, there are still many difficulties involved in designing a priority-rollback scheme to guarantee that no transactions block. Another, related difficulty in the design of a mechanism for allowing transactions to commit: even though the concurrency control guarantees eventual performance of all of the steps of a correct execution sequence e, it does not necessarily follow that the concurrency control can determine a particular point in time when each transaction can no longer have any of its steps rolled back! This is apparently a greater difficulty for multilevel atomicity than it is for ordinary atomicity, since multilevel atomicity allows (even if there are only a finite number of entities) an infinite chain of transactions $t_1, t_2, t_3, \ldots$ such that for each i_1 there are steps α of t_i and β of t_{i+1} with $\beta \underset{e}{\leq} \alpha$. This means that it is quite plausible that a rollback of steps of t_{i+1} can cause a rollback of steps of t_i, and so on.

10. Further Research

Here, I have really only suggested a new, general correctness criterion. It remains to design detailed concurrency controls based on this criterion, in order to determine if the generalization can be exploited for increased efficiency.

Acknowledgements. The author is grateful to Nancy Griffeth, Mike Fischer and Mike Merritt for many discussions about the subjects covered in this paper.

References

[BG] Bernstein, P.A. and Goodman, N. Fundamental Algorithms for Concurrency Control in Distributed Databases, Computer Corporation of America, Technical Report, February 1980.

[C] Clark, D. Oral Presentation at the Fifth Berkeley Workshop on Distributed Data Management and Computer Networks, February 3-5, 1981.

[EGLT] Eswaren, K.P., Gray, J.N., Lorie, R.A. and Traiger, I.L. The Notions of Consistency and Predicate Locks in a Database System, Communications of the ACM, Vol. 19, Number 11, pp. 624-633, November 1976.

[FGL] Fischer, M.J., Griffeth, N.D., and Lynch, N.A. Global States of a Distributed System, Proceedings of IEEE Symposium on Reliability in Distributed Software and Database Systems, July 1981.

[G] Garcia-Molina, H. Using Semantic Knowledge for Transaction Processing in a Distributed Database, Technical Report #285, Princeton University Department of Electrical Engineering and Computer Science, April 1981.

[GLPT] Gray, J.N., Lorie, R.A., Putzolu, G.R., and Traiger, L.I. Granularity of Locks and Degrees of Consistency in a Shared Data Base, Proceedings IFIP Working Conference on Modelling of Data Base Management Systems, Freudenstadt, Germany, pp. 695-723, January 1976. Also in Modelling in Data Base Management Systems, G.M. Nijssen (ed) North Holland Publishing Company, pp. 365-395, 1976.

[L] Lamport, L. Towards a Theory of Correctness of Multi-user Database Systems, Massachusetts Computer Associates, CA-7610-0712, October 1976.

[LF] Lynch, N.A., Fischer, M.J. On Describing the Behavior and Implementation of Distributed Systems, to appear in Theoretical Computer Science, Vol. 13, pp. 17-43, January 1981.

[RSL] Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M. System Level Concurrency Control for Distributed Database Systems, ACM Transactions on Database Systems, Vol. 3, No. 2, pp. 178-198, June 1978.